

АССЕМБЛЕР

- Пильщиков В.Н. Программирование на языке ассемблера IBM PC.- М.: "Диалог- МИФИ", 2005-2010, -288с.
- Юров В., Хорошенко С. ASSEMBLER. Учебный курс. – Спб., Питер, 1999
- Зубков С.В. Ассемблер для DOS, Windows, UNIX.,- М. ДМК, 2004
- Джордейн Р. Справочник программиста персонального компьютера фирмы IBM. М., 1991.
- Сван Т. Освоение Turbo Assembler. - Киев: Диалектика, 1996.- 544с.

Интернет:

www.citforum.ru, www.rusdoc.ru, emanual.ru,
www.kalashnikoff.ru, www.firststeps.ru,
www.codenet.ru, www.wasm.ru, www.intuit.ru



Использование двоичной и шестнадцатеричной систем счисления

d	b	h
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F
16	10000	10

$$150d = 10010110b = 96h$$

150d → b	Остаток
$150 / 2 = 75$	0 ↑
$75 / 2 = 37$	1
$37 / 2 = 18$	1
$18 / 2 = 9$	0
$9 / 2 = 4$	1
$4 / 2 = 2$	0
$2 / 2 = 1$	0
$1 / 2 = 0$	1

Результат: 10010110 b

7 6 5 4 3 2 1 0
10010110b → d
$2^7 + 2^4 + 2^2 + 2^1 = 150$
Результат: 150d

150d → h	Остаток
$150 / 16 = 9$	6 ↑
$9 / 16 = 0$	9
Результат: 96h	

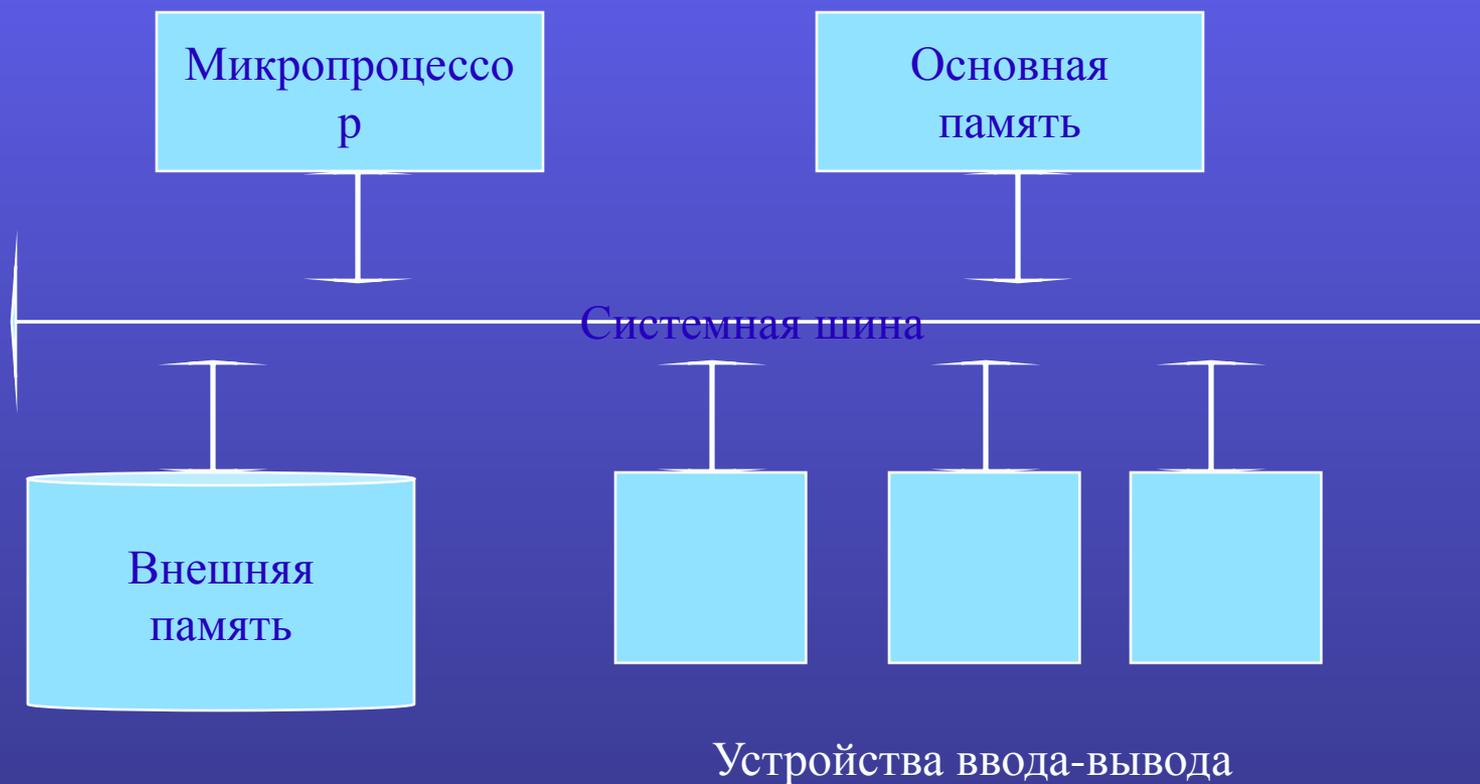
10
96h → d
$9 * 16^1 + 6 * 16^0 = 150$
Результат: 150d

10010110b → h
$ \begin{array}{c} 1001 \quad \uparrow \quad 0110 \\ \swarrow \quad \searrow \quad \swarrow \quad \searrow \\ 9 \quad \quad \quad 6 \end{array} $
Результат: 96h

Контрольное тестирование 1

1	Какое двоичное число следует за числом	11011b	
2	Какое двоичное число предшествует числу	110000b	
3	Какое шестнадцатеричное число следует за числом	1999h	
4	Какое шестнадцатеричное число следует за числом	9Fh	
5	Какое шестнадцатеричное число предшествует числу	A00h	
6	Вычислить в шестнадцатеричном формате	AABh-2	
7	Какое максимальное десятичное число можно представить заданным числом бит	15	
8	Сколько бит (минимум) потребуется для представления данного числа	17h	
9	Сколько байт потребуется для представления данного числа	79Eh	
10	Представьте десятичное число в формате b и h	99	
11	Представьте двоичное число в формате d и h	111110b	

Архитектура персонального компьютера



История развития микропроцессоров Intel (семейство «x86»)

Год выпуска	Тип МП	Тактовая частота, МГц	Разрядность данных	Разрядность адреса	Макс. объем памяти
1972	i8008	4	8	16	64 Кб
1978	i8086	16	16	20	1 Мб
1982	i80286	40	16	20	1 Мб
1985	i386	40	32	32	4 Гб
1989	i486	100	32	32	4 Гб
1993	Pentium	233	32	32	4 Гб
1997	Pentium II	700	32	32	4 Гб
2000	Pentium IV	>3000	32	32	4 Гб
2001	Itanium	>1000	64	64	16 Еб
2005	Pentium IV EM64T	>3000	64	64	16 Еб
2005	Pentium D	>2800	64	64	16 Еб
2008	Core i7	3200	64	64	16 Еб

Память ПК

Основная память



- ✓ Служит для размещения кода и данных программ в период выполнения
- ✓ ЭнергоЗАВИСИМА
- ✓ Возможен непосредственный доступ со стороны процессора с помощью команд (MOV)
- ✓ Высокая скорость доступа

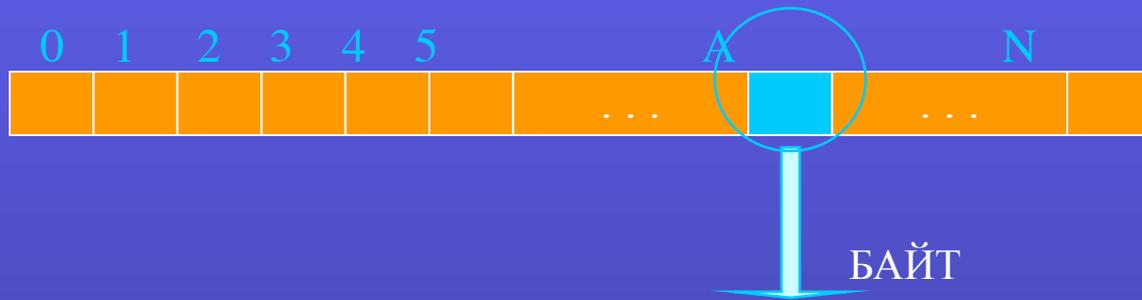
Внешняя память



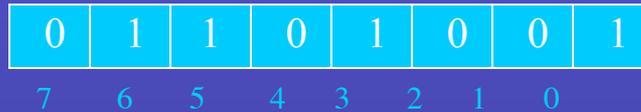
- ✓ Служит для долговременного хранения программ и данных в виде файлов
- ✓ ЭнергоНЕЗАВИСИМА
- ✓ Возможен опосредованный доступ со стороны процессора через интерфейс ввода-вывода (IN,OUT)
- ✓ Низкая скорость доступа



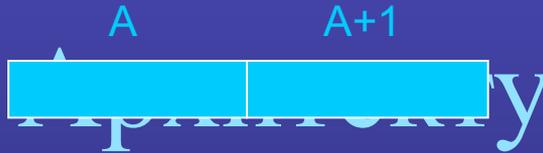
Основная память



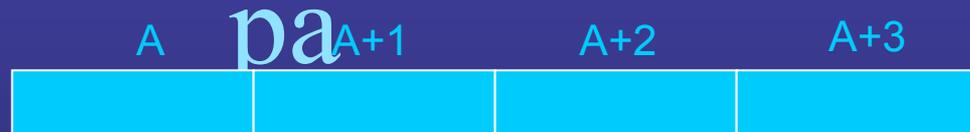
$$N_{\max} = \begin{cases} 2^{16} = 64\text{Кб} \\ 2^{32} = 4\text{Гб} \\ 2^{64} = 16\text{Еб} \end{cases}$$



СЛОВО



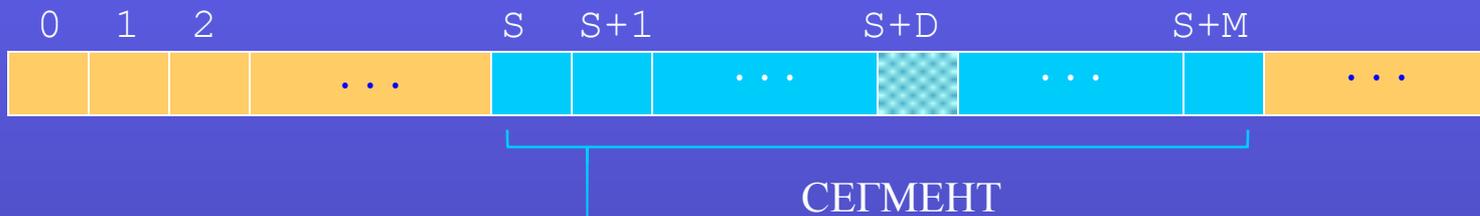
ДВОЙНОЕ СЛОВО



персональ

цого

Сегментированная модель памяти



S – абсолютный адрес сегмента

D – смещение ячейки внутри сегмента (эффективный адрес)

M – размер сегмента (≤ 64 Кб для i8086, ≤ 4 Гб для i80386)

A = S + D – абсолютный адрес ячейки внутри сегмента

P = S / 10h – позиция сегмента (номер параграфа)

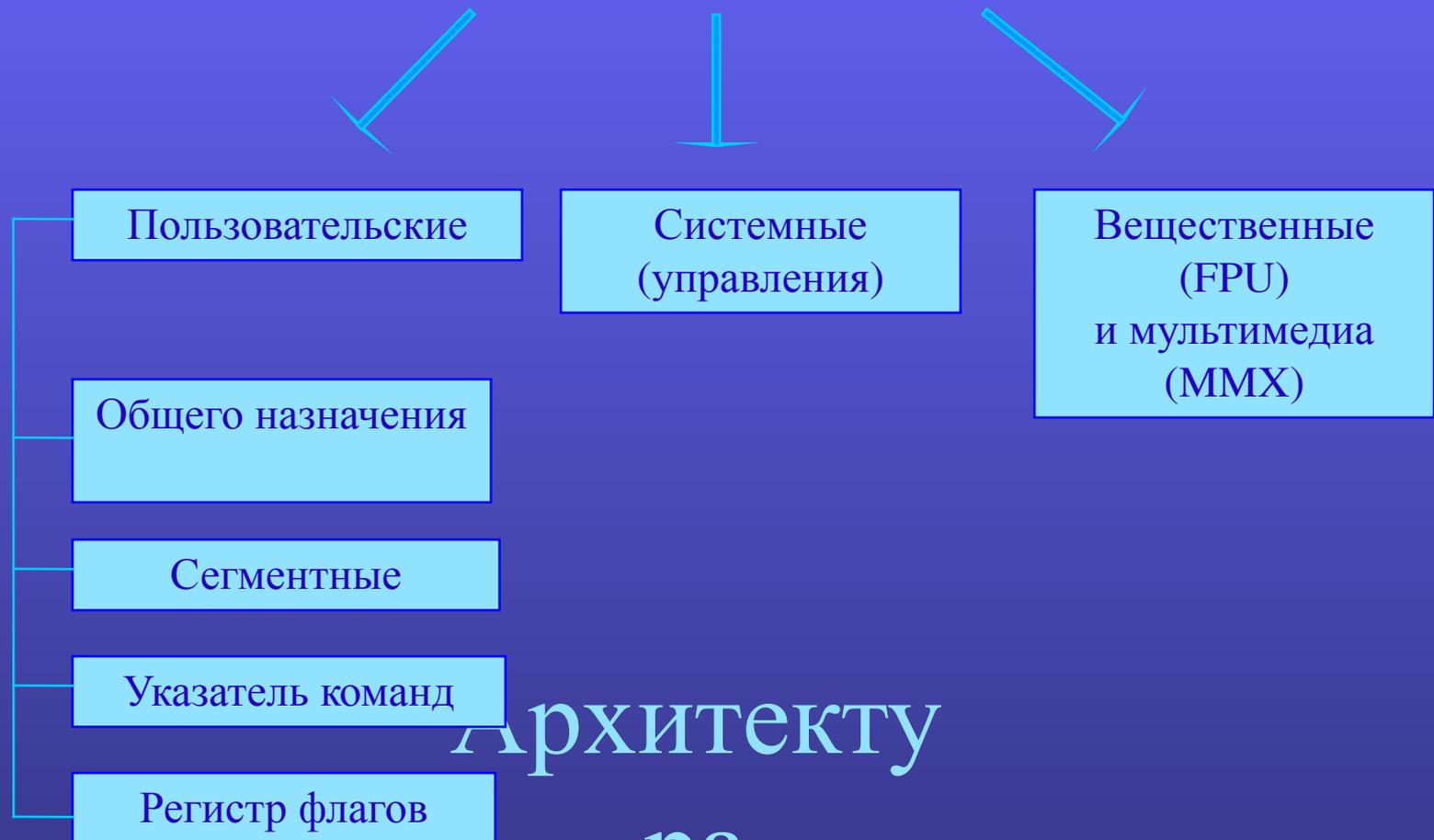
P : D – сегментированный адрес ячейки внутри сегмента

P	D	=>	+	48E30h	S=P*10h
48E3h	: 0027h			0027h	D
			<hr style="width: 50px; margin: 0 auto;"/>	48E57h	A

персональ

ного

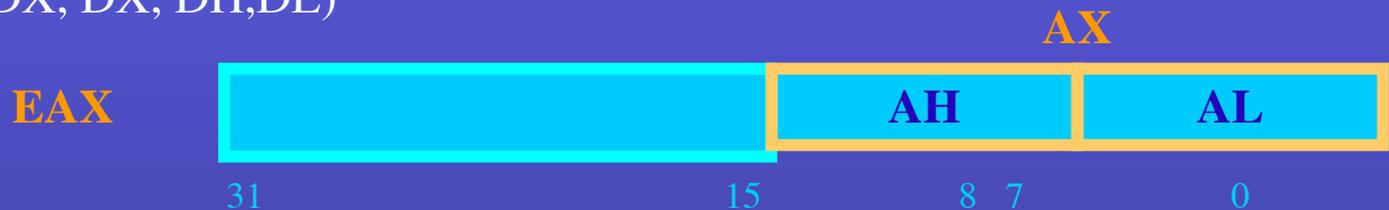
Регистры процессора



Архитекту
ра
персональ

Регистры общего назначения

- Аккумулятор (EAX, AX, AH, AL)
- База (EBX, BX, BH, BL)
- Счетчик (ECX, CX, CH, CL)
- Данные (EDX, DX, DH, DL)



- Индекс источника (ESI, SI)
- Индекс приемника (EDI, DI)
- Указатель базы (EBP, BP)
- Указатель стека (ESP, SP)



Архитекту

персональ

цого

10

Сегментные регистры

- Указатель сегмента кода CS
- Указатель сегмента стека SS
- Указатель сегмента данных DS
- Указатель дополнительного сегмента данных ES
- Указатель дополнительного сегмента данных GS
- Указатель дополнительного сегмента данных FS



ра

персональ

Регистр флагов



Флаги условий:

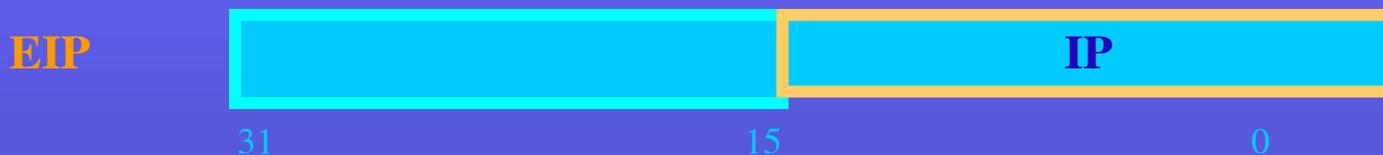
- CF – флаг переноса
- OF – флаг переполнения
- ZF – флаг нуля
- PF – флаг четности
- SF – флаг знака
- AF – флаг дополнительного переноса

Флаги состояний:

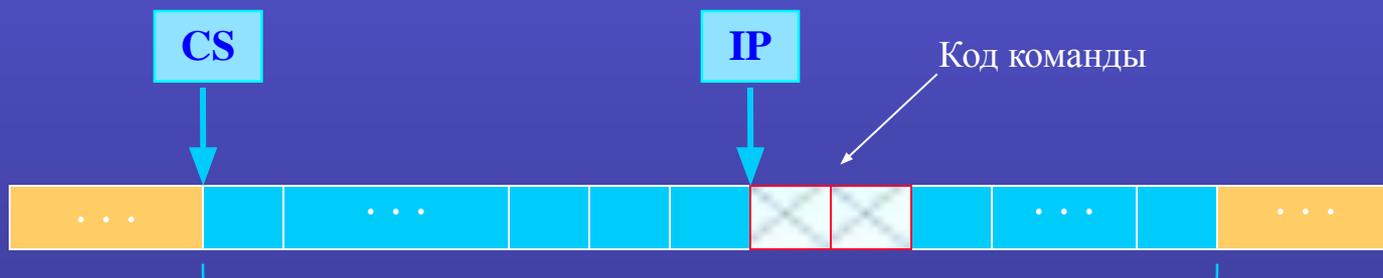
- TF – флаг трассировки
- IF – флаг прерывания
- DF – флаг направления

Архитектура
ра
персональ

Регистр указателя команд



CS:IP (CS:EIP) – адрес размещения в памяти следующей команды



Архитекту
ра

персональ

Регистры 64-разрядного процессора

- Регистры общего назначения (64 бита):

RAX, RBX, RCX, RDX, RSI, RDI, RBP, RSP, R8, R9, ..., R15



- Сегментные регистры: CS, DS, SS, ES, GS, FS (16 бит)

- Регистр указателя команд: RIP (64 бита)

- Регистр флагов: EFLAGS (32 бита)

Архитекту

ра

персональ

Алгоритм работы процессора



персональ

процессор

Инициализация счетчика команд CS:IP

Чтение и анализ кода команды

Чтение операндов



Формирование нового значения CS:IP

Выполнение команды

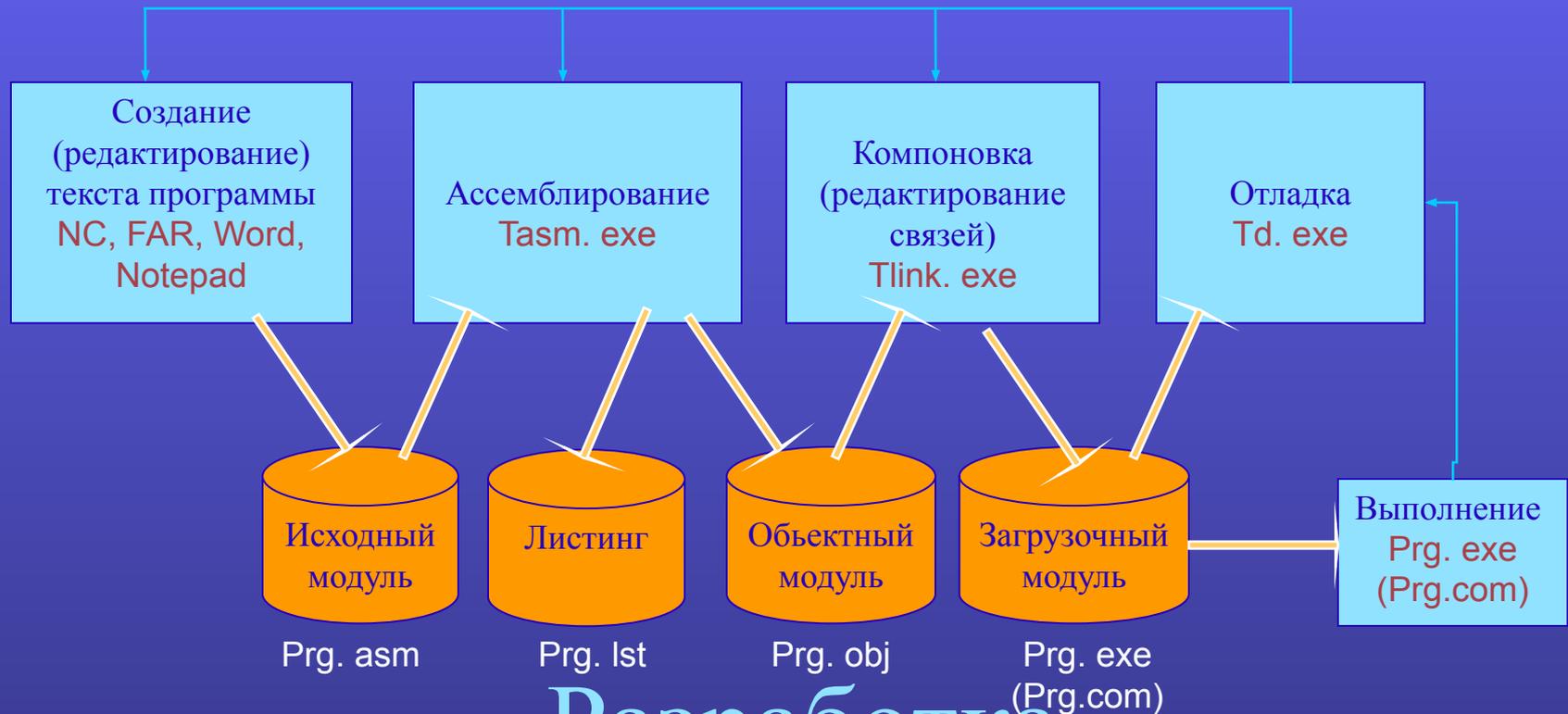
Формирование IP:=IP+Lc



38A20 38A21 38A22 38A23 38A24 38A25 38A26 38A27

38A2F 38A30 38A31 38A32

Этапы разработки программ на Ассемблере



Разработка
программ

на

17

Общий вид программы на Турбо Ассемблере

```
.MODEL модель; используемая модель памяти  
.STACK N ; сегмент стека (N – размер)  
.DATA ; сегмент данных  
...  
Описание данных программы  
...  
.CODE ; сегмент кода  
START: ; точка входа в программу  
...  
Код программы  
...  
END START ; конец сегмента кода
```

Размещение ехе-программы в памяти

DS:0	45 EF 38 69 A3 00 65 77 0D 00 12 34 FF 00 00 00	Сегмент данных
CS:0	34 56 00 75 AB 00 C5 D6 12 34 56 78 90 94 56 34 23 DD FF 5F 4A B3 CC 43 26 77 80 00 BB D1 2F E5 00 67 85 34 2A A4 BD FF 09 57 20 81 27 56 00 ED	Сегмент кода
SS:0	23 DD FF 5F 4A B3 CC 43 26 77 80 00 BB D1 2F E5 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	Сегмент стека

- Сегмент данных служит для размещения переменных программы и адресуется с помощью сегментного регистра DS.
- Сегмент кода предназначен для размещения машинного кода программы. Ячейки сегмента кода адресуется с помощью пары регистров CS:IP.
- Сегмент стека служит для временного хранения данных программы при вызове подпрограмм, обработке прерываний и т.п. Сегмент стека адресуется с помощью сегментного регистра SS.

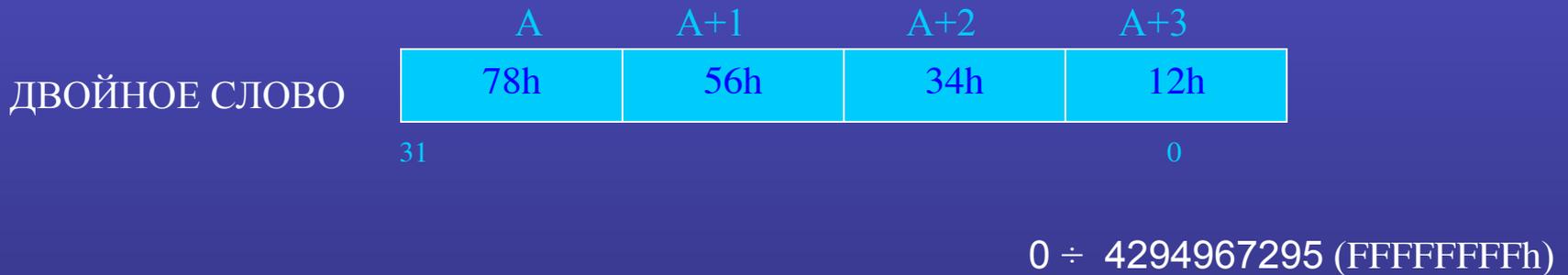
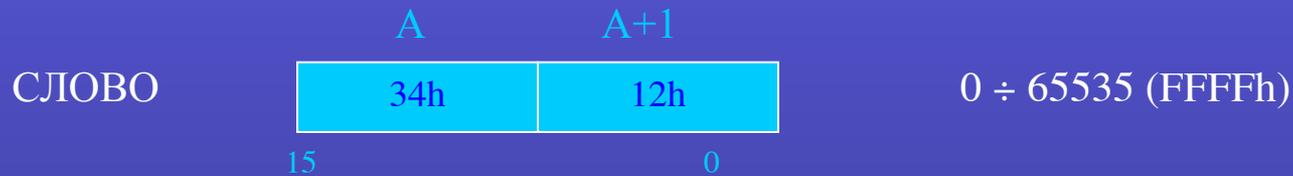
Разработка программ

Для микропроцессора i8086 размер сегмента не может превышать 64К

Представление данных в памяти

Целые числа без знака

Формат little-endian



Представл ение

Дополнительный код

$$\text{доп}(X) = \begin{cases} X, & \text{если } X \geq 0 \\ 2^k - |X|, & \text{если } X < 0 \end{cases}$$

□ Байт

98d → 62h → 01100010b

-98d → (256 - 98 = 158d) → 9Eh → 10011110b

□ Слово

-98d → (65536 - 98 = 65437) → FF9Eh → 1111111110011110b



Представл
ение

Целые числа со знаком



127d → 01111111b → 7Fh

-2d → 11111110b → FEh

1d → 00000001b → 01h

-127d → 10000001b → 9Eh

-1d → 11111111b → FFh

-128d → 10000000b → 80h

СЛОВО



-32768 ÷ 32767

ДВОЙНОЕ СЛОВО



Представл
ение

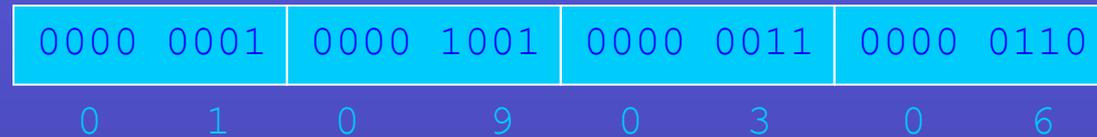
ение

22

Двоично-десятичные числа

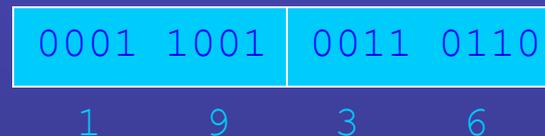
Неупакованный BCD формат

1936d →



Упакованный BCD формат

1936d →



Представл

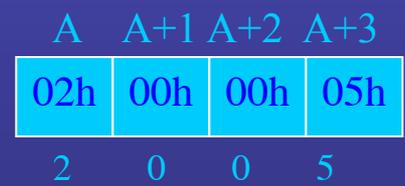
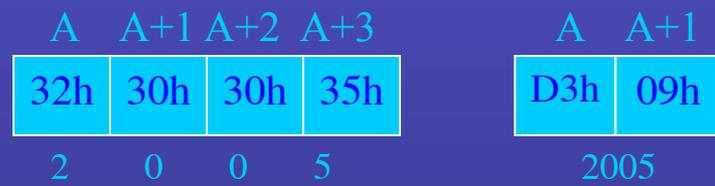
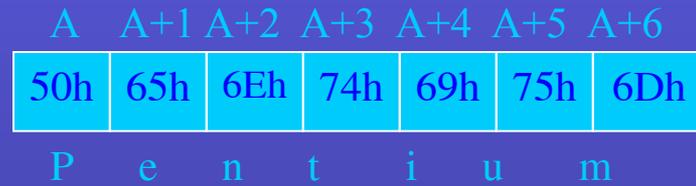
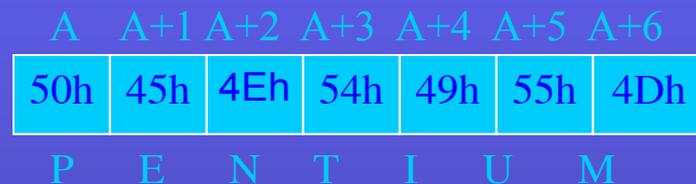
ение

23

Представление символов

Таблица кодировки символов (стандарт ASCII)

	0	1	2	3	4	5	6	7
0	NUL	DLE		0	@	P	`	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	\$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(8	H	X	h	x
9	HT	EM)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[k	{
C	FF	FS	,	<	L	\	l	
D	CR	GS	-	=	M]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	~

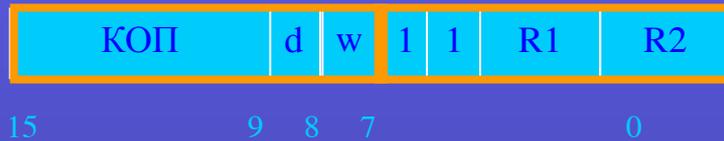


Представл

Представление команд (на примере двухоперандных команд)

КОП операнд1 , операнд2

1. Формат регистр – регистр (2 байта)



d – признак результата (1 - в R1, 0 - в R2)

w – тип операндов (1 – слово, 0 – байт)

```
MOV AX, BX    ; 10001011 111000011 = 8BC3h
ADD AX, BX    ; 00000011 111000011 = 03C3h
SUB AX, BX    ; 00101011 111000011 = 2BC3h
```

w=0	w=1	R1, R2
AL	AX	000
CL	CX	001
DL	DX	010
BL	BX	011
AH	SP	100
CH	BP	101
DH	SI	110
BH	DI	111

2. Формат регистр – память (2 - 4 байта)

```
MOV DX, HELLO
```

3. Формат регистр – непосредственный операнд (3 - 4 байта)

```
MOV BX, 5
```

4. Формат память – непосредственный операнд (4 - 6 байт)

```
MOV [SI], 01Ah.
```

Представл

Основные элементы языка Ассемблер

Алфавит

- латинские буквы **A – Z, a – z**
- цифры **0 – 9**
- знаки **? @ \$ _ &**
- разделители **, . [] () < > { } + / * % ! " \ \ = # ^**

Лексемы

- идентификаторы (имена)
- числа
- цепочки символов (строки)

Правила описания



Основные
ЭЛЕМЕНТЫ

Основные элементы языка Ассемблер

Идентификаторы

- Пслужебные слова (AX, EIP, MOV, ADD, END, SEGMENT);
- Пимена (PRIMER, MASSIV, CYCLE,).

Особенности использования идентификаторов:

- Может включать латинские буквы, цифры, а также специальные символы ? . @ \$ _
- Идентификатор не может начинаться с цифры
- Длина имени может быть любой, но значащими являются только первые 31 символов
- Пробелы внутри идентификатора не допустимы
- Если используется точка в имени, то она может стоять только на первой позиции (.A)
- Допускается применение как прописных так и строчных букв (AX, Ax, ax).

Основные элементы

Основные элементы языка Ассемблер

Целые числа

- Десятичные: 15, -3, 123d (пробелы недопустимы);
- Двоичные: 1011b, 1000B;
- Восьмеричные: 127q, 345o (состоят из цифр 0÷7);
- Шестнадцатеричные: 1234h, 0ABCh, 1d8ff7h, (AF5h – неправильно)

Цепочки символов (строки)

Любая последовательность символов алфавита языка, заключенная в кавычки “...” или апострофы ‘...’

“Дата рождения ‘05.05.99’”

‘A + B’

“Assembler”

“X’ – неправильно

Основные
элементы

Структура программы на языке Ассемблер

```
предложение  
предложение  
  
.  
.  
.  
  
предложение
```

Типы предложений

- Команды;
- Макрокоманды;
- Директивы;
- Комментарии.

- Каждое предложение в отдельной строке
- Длина предложения не более 131 символа
- Переносы не допускаются

ОСНОВНЫЕ
ЭЛЕМЕНТЫ

Команды

[метка:] мнемокод [операнды] [; комментарий]

метка: – служит для переходов на данную команду;

мнемокод – служебное слово, указывающее операцию, которую надлежит выполнить (ADD, SUB, JMP)

операнды – аргументы, над которыми выполняется операция, определенная мнемокодом. В качестве операндов можно использовать числа, цепочки символов, служебные слова, выражения и операторы.

комментарий – служит для пояснения действия команды

Основные типы операндов

1. *Непосредственный* - указывается в самой команде в виде числового или символьного значения (i8, i16, i32):

```
MOV AL, 5
```

2. *Регистровый* - задается через соответствующий регистр микропроцессора (AL, DS, ESI) (r8, r16, r32, sr) :

```
MOV AX, BX
```

3. *Операнд в памяти* - в команде указывается имя операнда или адрес ячейки памяти (m8, m16, m32):

```
MOV DX, NAME  
MOV ES: [DI], 118
```

Операнды могут задаваться явно и неявно:

Явный - присутствует в команде в виде имени, значения или выражения;

Неявный - не присутствует в команде, но подразумевается.

Основные
ЭЛЕМЕНТЫ

Прочие типы предложений языка

Директивы

- предложения, содержащие символическое указание ассемблеру (не преобразуются в машинный код)

```
.MODEL SMALL
```

```
.CODE
```

```
X DB 17 h
```

```
ORG 100h
```

Макрокоманды

- предложения, которые в процессе ассемблирования замещаются другими предложениями ассемблера.

```
OUT_STR "Hello"
```

Комментарий

- предложения, служащие для пояснения текста программы (игнорируются ассемблером)

```
MOV DS, AX ; комментарий  
COMMENT * многострочный  
           комментарий *
```

**ОСНОВНЫЕ
ЭЛЕМЕНТЫ**

Определение данных Ассемблера

[имя] директива операнд { [,операнд] }

Директива :

DB – байт, **DW** – слово (2 байта), **DD** – двойное слово (4 байта),

DF – 6 байт, **DP** – 6 байт, **DQ** – 8 байт, **DT** – 10 байт

Операнды:

- число;
- строка (цепочка символов);
- неопределенное значение (?);
- повторитель (DUP);
- константа;
- константное выражение;
- адресное выражение;

Имя:

- интерпретируется как **адрес памяти** для ссылки на ячейки размещения данных;
- характеризуется **типом** (TYPE), т.е. количеством занимаемых ячеек памяти;
- определяет **значение**, т.е. данные в памяти (необязательно).

Определен
ие данных
Ассемблер

Директива DB: определение байта

[имя] DB операнд {[,операнд]}

TYPE = 1, диапазон значений: -128.. 255

```
.DATA          ; сегмент данных
X   DB   ?
Y   DB   175      ; AFh
ZET  DB   0Ch
      SYM   DB   '*' ; 2Ah
      DB   5
      N2   DB   -3   ; FDh
T   DB   TYPE X
```



ие данных

Ассемблер

Определение массивов и строк

```
MAS DB 1,2,5,9,0,6
```

MAS	MAS+1	MAS+2	MAS+3	MAS+4	MAS+5
01	02	05	09	00	06

```
STR DB 'a', 'b', 'c'  
или  
STR DB 'abc'
```

'a'	'b'	'c'
-----	-----	-----

```
K DB 1, 1, 1, 1, 1  
или  
K DB 5 DUP (1)
```

01	01	01	01	01
----	----	----	----	----

```
L DB 5, 3 DUP (0,2 DUP (?))
```

05	00	?	?	00	?	?	00	?	?
----	----	---	---	----	---	---	----	---	---

```
BCD DB 1,8,7,3
```

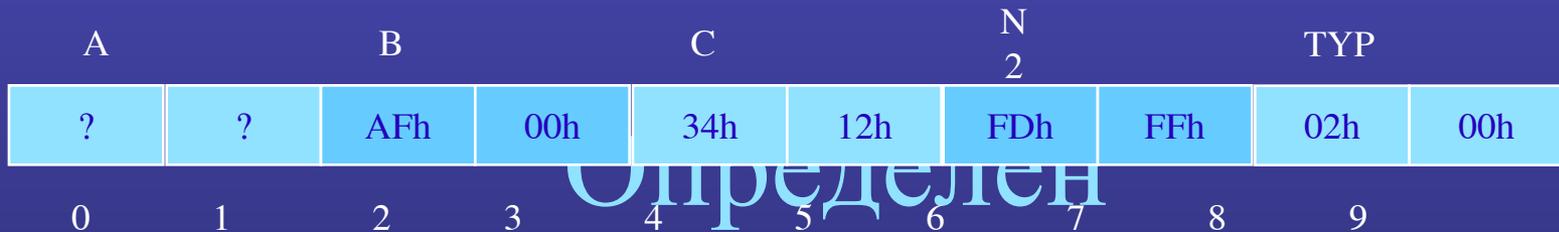
01	08	07	03
----	----	----	----

Директива DW: определение слова

```
[имя] DW операнд { [,операнд] }
```

TYPE = 2, диапазон значений: -32768.. 65535

```
.DATA ; сегмент данных
A DW ?
B DW 175 ; 00AFh
C DW 1234h
N2 DW -3 ; FFFDh
TYP DW TYPE A
```



Директива DD: определение двойного слова

```
[имя] DD операнд { [,операнд] }
```

TYPE = 4, диапазон значений: -214783648 .. 4294967295

```
.DATA ; сегмент данных  
A DD ?  
B DD 123456h  
    TYPE DD TYPE A
```

A				B				TYPE			
?	?	?	?	56h	34h	12h	00h	04h	00h	00h	00h
0	1	2	3	4	5	6	7	8	9	10	11

Определен
ие данных
Ассемблер

Константы. Директива эквивалентности

имя EQU операнд

(имя := операнд)

- ✓ все вхождения имени константы в программе ассемблер заменяет на значение операнда;
- ✓ памяти для размещения константы не выделяется

```
PLUS EQU '+'  
.  
.  
P DB PLUS
```

```
N EQU 100  
.  
.  
X DB N DUP (?)
```

```
REZ EQU X*(Y+2)  
.  
.  
ADD AX, REZ
```

Директива присваивания

имя = операнд

```
K = 10  
.  
.  
A DB K  
.  
.  
MOV AX, K
```

ен
ие данных
Ассемблер

Выражения

Выражением называется синтаксическая конструкция, которая содержит числа, имена констант и переменных, а также *операторы*, определяющие действия над элементами выражений.

```
Z DB (3*Y+X)/2-486
MOV AX,OFFSET MAS
MOV AL,BYTE PTR [DI]+1
```

Выражения вычисляются во время ассемблирования, поэтому не могут включать величины, хранящиеся в регистрах или в памяти.

Операторы

арифметические операторы:

+, -, *, / - сложение, вычитание, умножение, деление;
MOD – остаток от деления;
() – порядок действий;

логические операторы:

NOT (нет), AND (и), OR (или), XOR (исключающее или), SHL, SHR (сдвиг);
EQ, NE, LT, LE, GT, GE – логические условия;

прочие операторы:

:, [], PTR, OFFSET, SEG, TYPE, HIGH, LOW, SHORT, LENGTH, SIZE и др.

Определен
ие данных
Ассемблер

Константные выражения

- ✓ включают числа, константы и символы
- ✓ значение константного выражения есть целое число

```
X    DW    5*8-24                ;X= 16
```

```
K    EQU   3                    ;K = 3  
Y    DB    (3*K-1)/2 DUP(?)     ;Y = ?, ?, ?, ?
```

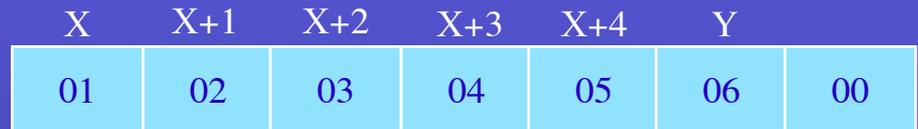
```
t_size EQU 80  
e_size EQU 2  
.  
.  
.  
MOV CX, t_size/e_size          ;CX = 40
```

Определен
ие данных
Ассемблер

Адресные выражения

- ✓ включают числа, метки, имена переменных, а также счетчик размещения \$
- ✓ значением адресного выражения является 16-и битовое или 32-х битовое целое число, интерпретируемое как адрес.

```
.DATA
X  DB 1,2,3,4,5
Y  DW 6
. . .
MOV AL,X+3    ;AL:= 4
MOV BL,X+5    ;BL:= 6
MOV CX,Y-1    ;CX:= 0605h
```



```
.DATA
A  DB 8
X  DB 40 DUP(?)
SIZE_X EQU ($-X)/TYPE X
```



Определен
ие данных
Ассемблер

Использование оператора PTR

BYTE PTR
WORD PTR
DWORD PTR

```
MOV [BX],28h ; ?
```

;Неизвестно сколько ячеек памяти нужно использовать: (2,4,8) ?

```
MOV [BX],BYTE PTR 28h ; пересылка одного байта по адресу (BX)
```

```
MOV BYTE PTR [BX],28h ; то же
```

```
MOV [BX],WORD PTR 28h ; пересылка слова по адресу (BX)
```

```
Z DD 123456h
```

```
MOV BYTE PTR Z,0
```

```
MOV BYTE PTR Z+1,0
```

```
MOV WORD PTR Z,3456h
```

Z

56	34	12	00
----	----	----	----

00	34	12	00
----	----	----	----

00	00	12	00
----	----	----	----

56	34	12	00
----	----	----	----

Команды

пересылки

41

Контрольное тестирование 2

```

.DATA
A1  DB  17
A2  DW  2  DUP  (0AE5h)
A3  DB  -1, -3, 10B, ?
A4  DW  15D, 0, -2, TYPE A1
N    EQU  ($-A4)/TYPE A4
A5  DB  'LEO'
A6  DD  123456H
A7  DW  A5
A8  DB  N  DUP (252)
    
```

1. Покажите, как представлена каждая переменная в памяти побайтно (в hex)
2. Чему равно значение объявленных констант?
3. Сколько байт памяти и какие значения адресуются с помощью следующих выражений: A2, A3-2, A3+2, A4+3, A7-2, A7, A8[2]



Разделение
данных

Ассемблер

Основные команды языка Ассемблер



Команды
языка

Команды пересылки

- ❑ Общего назначения
MOV, XCHG
- ❑ Команды чтения / записи в стек
POP, PUSH
- ❑ Команды для ввода / вывода
IN, OUT
- ❑ Команда для пересылки адресов
LEA, LDS, LES, ...

Команды

языка

Ассемблер

Команда пересылки MOV

MOV приемник, источник
(**приемник := источник**)

Ограничения по использованию операндов:

- Операнды должны быть согласованы по размеру (типу): 8,16,32;
- Нельзя пересылать из памяти в память;
- Приемник не может быть непосредственным операндом и регистром CS;
- Нельзя пересылать из сегментного регистра в сегментный;
- Нельзя пересылать непосредственный операнд в сегментный регистр;
- Нельзя пересылать из памяти в сегментный регистр;

```
MOV AL, 8           ; регистр :=непосредственный операнд
MOV DS, AX          ; регистр := регистр
MOV AX, MEM+1       ; регистр := память
MOV MEM, DX         ; память := регистр
```

; Недопустимые команды:

```
MOV DS, 8           ; запись в сегм.регистр непоср. значения
MOV CH, 283         ; (CH) <= 255 (1 байт)
MOV MEM, MASSIV     ; пересылка из памяти в память
```

КОМАНДЫ

пересылки

Методы адресации (способы задания операндов в памяти)

Прямая адресация

```
MOV AX, MEM ; AX := (DS:MEM) () - содержимое памяти или регистра  
; смещение - по размещению MEM в сегменте данных  
MOV AX, ES:MEM ; сегментная часть адреса определена явно
```

Абсолютная адресация

```
MOV AX, ES:0005  
; AX := (ES:0005)
```

Косвенная базовая

```
MOV AX, [BX] ; AX := (DS: (BX) )  
; содержимое BX интерпретируется как адрес памяти  
; в i8086 для базирования используют BX, SI, DI, BP
```

Косвенная базовая адресация со смещением

```
MOV AX, MAS[BX] ; AX := (DS:MAS + (BX) )  
MOV AX, [BX+2] ; AX := (DS: (BX) + 2)  
MOV AX, [BX]+2 ; AX := (DS: (BX) + 2)
```

Методы адресации (способы задания операндов в памяти)

Косвенная базовая индексная адресация

```
MOV AX, [BX] [SI] ; AX := (DS: (BX) + (SI))
MOV AX, MAS [BX] [SI] ; AX := (DS: MAS + (BX) + (SI))
; BX-база, SI-индекс
; Допустимые регистры:
; Для i8086: (BX, SI), (BX, DI), (BP, SI), (BP, DI)
; Для i386: любые POH
```

Косвенная базовая индексная адресация со смещением

```
MOV AX, MAS [BX] [SI] + 2 ; AX := (DS: MAS + (BX) + (SI) + 2)
MOV AX, MAS [BX] [SI + 2] ; AX := (DS: MAS + (BX) + (SI) + 2)
```

Косвенная адресация с масштабированием

```
MOV AX, MAS [ESI * 2] ; AX := (DS: MAS + (ESI) * 2)
```

Команда обмена значениями XCHG

```
XCHG  операнд1 , операнд2  
      В := операнд1  
      операнд1 := операнд2  
      операнд2 := В
```

- Операнды должны быть согласованы по размеру (типу): 8,16,32
- Оба операнда не могут быть ячейками памяти одновременно
- Исключается использование непосредственных операндов

```
XCHG  AX, BX  
XCHG  CX, MEM
```

Команда загрузки адреса LEA

```
LEA  операнд1 , операнд2  
операнд1 := Адрес (операнд2)
```

Пересылает эффективный адрес переменной (смещение в сегменте данных) в регистр (r16)

```
MAS  DB  10 dup (0)  
...  
LEA  BX, MAS ; в регистр BX записываем адрес переменной MAS  
MOV  DH, MAS ; в регистр DH записываем содержимое ячейки памяти
```

Арифметические команды

- ❑ Команды двоичной арифметики
ADD, SUB, MUL, IMUL, DIV, IDIV, INC, DEC, NEG
- ❑ Команды десятичной арифметики
AAA, AAS, AAM, AAD, ...
- ❑ Команды преобразования типов данных
CBW, CWD, ...

Арифмети
ческие

Команда сложения ADD

ADD **приемник, источник**
приемник := приемник + источник

- Операнды должны быть согласованы по типу (8, 16, 32)
- Операнды одновременно не могут быть ячейками памяти
- Приемник не может быть непосредственным операндом

```
ADD  AX, BX      ; AX := AX + BX
ADD  MEM, 28h    ; MEM := MEM + 28h
ADD  CX, BUFF    ; CX := CX + BUFF
```

Устанавливаемые флаги: ZF, SF, CF, OF, AF, PF

- Флаг переноса CF устанавливается, если есть перенос из старшего разряда
- Флаг переполнения OF устанавливается, если имеет место только один из двух переносов: из знакового (старшего) разряда либо в знаковый (старший) разряд
- Флаг нуля ZF устанавливается, если результат равен нулю
- Флаг знака SF устанавливается, если знаковый (старший) бит результата равен 1

```
MOV  AL, 1      ;
ADD  AL, -1     ; 00000001 + 11111111 = 00000000
                        ; CF=1, OF=0, ZF=1, SF=0
```

```
+ 00000001
  11111111
  -----
  00000000
  ~~~~~
```

ЧЕСКИЕ

Сложение беззнаковых чисел

- Если результат сложения превосходит по размеру приемник, то из полученного результата будет вычтено 2^k и этот результат будет записан в приемник (k - разрядность приемника)

$$\text{сумма}(X,Y) = \begin{cases} X+Y, & \text{если } X+Y < 2^k, \text{ CF}=0 \\ X+Y-2^k, & \text{если } X+Y \geq 2^k, \text{ CF}=1 \end{cases}$$

- Признаком правильности результата сложения беззнаковых чисел является значения флага переноса $\text{CF}=0$ ($\text{CF}=1$ – ошибка)

```
MOV AL, 254
ADD AL, 5 ; AL = 259 - 256 = 3, CF=1
```

```
  11111110
+ 00000101
-----
  1 00000011
```

CF

Арифмети
ческие

51

Сложение чисел со знаком

- Сложение чисел со знаком и без знака производится по одному алгоритму!
- Отрицательные числа представляются в дополнительном коде
- Признаком правильности результата сложения чисел со знаком является значения флага переполнения OF (OF=1 – ошибка)

$3 + (-1) = 2$

$$\begin{array}{r} 00000011 \\ + 11111111 \\ \hline 100000010 \end{array}$$

CF=1, OF=0

$-3 + 1 = -2$

$$\begin{array}{r} 11111101 \\ + 00000001 \\ \hline 11111110 \end{array}$$

CF=0, OF=0

$127 + 2 = -127 ?$

$$\begin{array}{r} 01111111 \\ + 00000010 \\ \hline 10000001 \end{array}$$

CF=0, OF=1

$-128 + (-1) = 127 ?$

$$\begin{array}{r} 10000000 \\ + 11111111 \\ \hline 101111111 \end{array}$$

CF=1, OF=1

Результат правильный

Результат неправильный

Арифмети
ческие

Команда вычитания SUB

SUB приемник , источник
приемник :=приемник-источник

```
SUB    AX, 17            ; AX :=AX - 17  
SUB    MEM, BX          ; MEM:=MEM - BX  
SUB    BUFF, 24513      ; BUFF:=BUFF - 24513
```

Устанавливаемые флаги: ZF, SF, CF, OF, AF, PF

- Флаг переноса CF устанавливается, если при вычислении старшего разряда результата был выполнен заем (т.е. если приемник меньше источника)
- Флаг переполнения OF устанавливается, если имеет место только один из двух заемов: из знакового (старшего) разряда либо в знаковый (старший) разряд (т.е. если вычитаются числа разных знаков и результат находится вне диапазона представления знаковых чисел)

Вычитание беззнаковых и знаковых чисел

- Вычитание знаковых и беззнаковых чисел производится по общему алгоритму: если приемник меньше источника, то к приемнику приписывается еще один единичный разряд слева (добавляется 2^k) и из полученного значения вычитается источник (k - разрядность приемника)

$$\text{разность}(X, Y) = \begin{cases} X - Y, & \text{если } X \geq Y, CF = 0 \\ (2^k + X) - Y, & \text{если } X < Y, CF = 1 \end{cases}$$

- Признаком правильности результата вычитания беззнаковых чисел является значения флага переноса CF (CF=1 – ошибка)
- Признаком правильности результата вычитания знаковых чисел является значения флага переполнения OF (OF=1 – ошибка)

```
  101011011
- 111011111
-----
  01001100
```

Беззнаковые: $91 - 239 = \#$, CF=1 – ошибка

Знаковые: $91 - (-17) = 108$, OF=0 – верно

Арифмети

ческие

54

Команды инкремента и декремента INC, DEC

Инкремент

INC операнд
операнд := операнд + 1

Декремент

DEC операнд
операнд := операнд - 1

```
INC AX          ; AX := AX + 1
INC MEM         ; MEM := MEM + 1
DEC AX          ; AX := AX - 1
DEC MEM         ; MEM := MEM - 1
```

Устанавливаемые флаги: OF, ZF (CF – не изменяется)

Арифмети
ческие

Команда инверсии знака NEG

NEG операнд
операнд := -операнд

```
MOV AH,1      ; 00000001b  
NEG AH       ; 11111111b = -1d
```

Устанавливаемые флаги:

- если операнд = 0, то CF = 0 ;
- если операнд ≠ 0, то CF = 1;
- если операнд = -128 или -32768, то OF = 1.

Команды умножения MUL, IMUL

MUL операнд (для беззнаковых чисел)
IMUL операнд (для чисел со знаком)

Тип операнда (1-й множитель)	2-й множитель	Произведение	Максимальное значение произведения	Граница для установки флагов CF и OF
r8, m8	AL	AX := операнд * AL	65535	255
r16, m16	AX	(DX, AX) := операнд * AX	4294967295	65535
r32, m32	EAX	(EDX, EAX) := операнд * EAX	$18,45 \cdot 10^{18}$	4294967295

Устанавливаемые флаги: CF, OF, ZF, SF

```

; ПРИМЕР 1: 100*2 = ?
MOV AL, 100
MOV BL, 2
MUL BL ; AX := AL * BL (100*2=200 <=255) → CF=OF=0 → произведение в AL
; ПРИМЕР 2: 100*3 = ?
MOV AL, 100
MOV BL, 3
MUL BL ; AX := AL * BL (100*3=300 >255) → CF=OF=1
; ПРИМЕР 3: 300*3 = ?
MOV AX, 300
MOV BX, 3
MUL BX ; (DX, AX) := AX * BX (300*3=900 <=65535) → CF=OF=0 → произведение в

```

AX

Команды деления DIV, IDIV

DIV операнд (для беззнаковых чисел)

IDIV операнд (для чисел со знаком)

Делимое	Делитель (операнд)	Частное	Остаток
AX	r8, m8	AL := AX / операнд	AH
(DX, AX)	r16, m16	AX := (DX, AX) / операнд	DX
(EDX, EAX)	r32, m32	EAX := (EDX, EAX) / операнд	EDX

- Флаги не устанавливаются
- Если операнд равен 0, или частное не уместится в соответствующем регистре, формируется прерывание (исключение)

```
MOV    AX, 301
MOV    BL, 2
DIV    BL        ; AX/BL => AL=150, AH=1
;
MOV    AX, 601
DIV    BL        ; AX/BL >255 => прерывание "divide overflow error"
;
MOV    AX, 601
MOV    DX, 0
MOV    BX, 2
DIV    BX        ; (DX, AX) / BX => AX=300, DX=1
```

Команды преобразования типов данных CBW, CWD

Преобразование байта в слово

CBW
AL=>AX

```
MOV AL, 5          ; 00000101b = 05h
CBW                ; AX:= 00000000000000101b = 0005h
MOV AL, -5        ; 11111011b = 0FBh
CBW                ; AX:= 1111111111111011b = 0FFFBh
;
MOV AL, 17        ; первое слагаемое - байт
MOV BX, 1073      ; второе слагаемое - слово
CBW                ; преобразование байта в слово
ADD BX, AX        ; осуществляем сложение
```

Если в AL беззнаковое число, то вместо CBW следует использовать MOV AH,0 !!!

Преобразование слова в двойное слово

CWD
AX=>(DX, AX)

```
MOV AX, 25        ; слово
MOV BX, 4         ; слово
CWD                ; преобразование слова AX в два слова (DX, AX)
DIV BX            ; делим (DX, AX) / BX
```

Команды передачи управления

- Команда безусловного перехода

JMP

- Команды условного перехода

**JE, JNE, JG, JGE, JL, JLE, JA, JAE, JB, JBE
JZ, JNZ, JC, JNC, JO, JNO, JS, JNS, JP, JNP
JCXZ, JCXNZ**

- Команды организации циклов

LOOP, LOOPE, LOOPNE

- Команды перехода с возвратом

CALL, RET, INT, IRET

Команды
передачи
управления

Переходы и метки

КОП метка

Команды перехода изменяют содержимое пары регистров CS : IP в соответствии со значением, определяемым операндом 'метка'

Метка характеризуется:

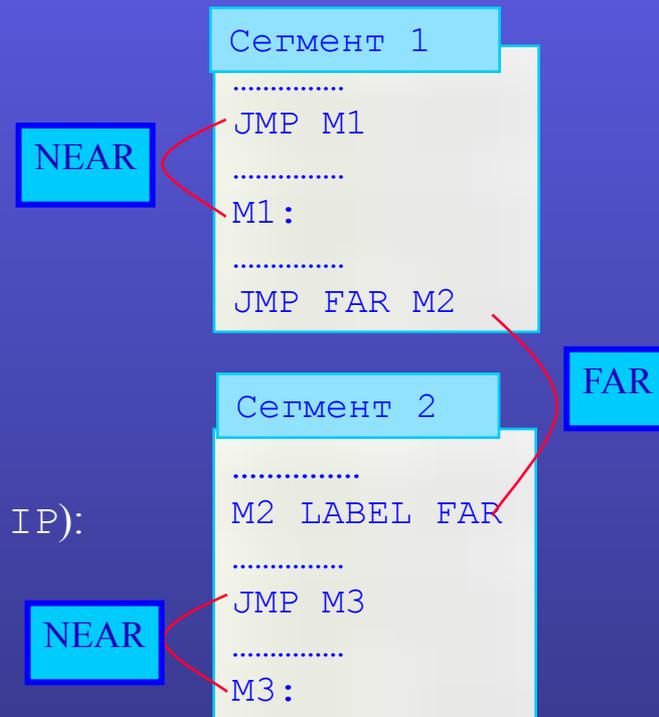
- 1) сегментом кода, в котором она описана;
- 2) смещением от начала сегмента кода;
- 3) атрибутом NEAR или FAR (ближний и дальний тип метки)

Типы переходов

NEAR – переход в пределах сегмента (изменяет только регистр IP):

- SHORT – короткий переход (в пределах: $-128 \div +127$ б)
- LONG – длинный переход (в пределах: $-32768 \div +32767$ б)

FAR – межсегментный переход (изменяет регистры IP и CS)



Команды
передачи

управления

Безусловный переход

JMP [SHORT] адрес (IP:=offset адрес)
JMP FAR PTR адрес (CS:=seg адрес, IP:=offset адрес)

Ближний (NEAR) переход

а) Прямой длинный переход вперед

```
JMP M1
. . .
M1: MOV AX, BX
```

б) Прямой короткий переход вперед

```
JMP SHORT M1
. . .
M1: MOV AX, BX
```

в) Прямой переход назад

```
M1: MOV AX, BX
. . .
JMP M1
```

г) Косвенный

```
M1: MOV AX, BX
. . .
LEA DX, M1
JMP DX
```

Дальний (FAR) переход

```
JMP FAR PTR M1
. . .
M1 LABEL FAR
MOV AX, BX
```

Условный переход

Jxx адрес
(IF условие xx THEN GOTO адрес)

- Переход по результату сравнения двух чисел (например по команде CMP)
- Переход по результату проверки состояния флагов
- Переход по содержимому регистра CX (ECX)

Тип перехода для i8086 near short, для i386: near long, near short

Команда сравнения

CMR операнд1, операнд2
(операнд1-операнд2 -> флаги)

- Флаги формируются так же, как и у команды вычитания SUB

Команды
передачи
управления

Переход по результату сравнения двух чисел

Команда перехода	СМР Оп1,Оп2	Типы операндов	Флаги
JE	Оп1 = Оп2	любые	ZF = 1
JNE	Оп1 ≠ Оп2		ZF = 0
JL (JNGE)	Оп1 < Оп2	со знаком	SF ≠ OF
JLE (JNG)	Оп1 ≤ Оп2		SF ≠ OF или ZF = 1
JG (JNLE)	Оп1 > Оп2		SF = OF и ZF = 1
JGE (JNL)	Оп1 ≥ Оп2		SF = OF
JB (JNAE)	Оп1 < Оп2	без знака	CF = 1
JBE (JNA)	Оп1 ≤ Оп2		CF = 1 или ZF = 1
JA (JNBE)	Оп1 > Оп2		CF = 0 и ZF = 0
JAE (JNB)	Оп1 ≥ Оп2		CF = OF

```

;вычислить Z = max(x, y)
;X, Y, Z - числа со знаком типа byte
MOV AL, X          ;AL:=X
CMP AL, Y          ;X - Y = ?
JGE M              ;перейти к метке M, если X >= Y
MOV AL, Y          ;выполнить AL:=Y, если X < Y
M : MOV Z, AL      ;Z:=AL
    
```

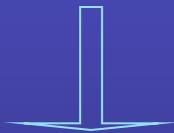
передачи

управления

64

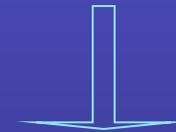
Переход по результату сравнения двух чисел (2)

```
;вычислить Z = max(x,y)
X   DB  11111111b
Y   DB  0
Z   DB  ?
...
    MOV  AL,X
    CMP  AL,Y
    JGE M
    MOV  AL,Y
M : MOV  Z,AL
```



Z=

```
;вычислить Z = max(x,y)
X   DB  11111111b
Y   DB  0
Z   DB  ?
...
    MOV  AL,X
    CMP  AL,Y
    JAE M
    MOV  AL,Y
M : MOV  Z,AL
```



Z=

Команды
передачи
управления

Переход по результату проверки флагов

Команда перехода	Условие перехода	Команда перехода	Условие перехода
JZ	ZF=1	JNZ	ZF=0
JS	SF=1	JNS	SF=0
JC	CF=1	JNC	CF=0
JO	OF=1	JNO	OF=0
JP	PF=1	JNP	PF=0

```

;C:=A*A+B (A,B,C-байты б/з)
MOV  AL,A
MUL  AL
JC   ERROR ;если A*A>255
ADD  AL,B
JC   ERROR ;если A*A+B>255
MOV  C,AL
...
ERROR:
    
```

Переход по содержимому регистра CX (ECX)

Команда перехода	Условие перехода	Команда перехода	Условие перехода
JCXZ	CX=0	JCXNZ	CX<>0

КОМАНДЫ
передачи
управления

Примеры организации ветвления программ

```
if X > 0 then  
  <блок 1>  
else  
  <блок 2>
```

```
CMP X, 0  
JLE M  
  <блок 1>  
JMP FIN  
M: <блок 2>  
FIN: ...
```

```
while X > 0 do  
  <блок>
```

```
BEG: CMP X, 0  
JLE FIN  
  <блок>  
JMP BEG  
FIN: ...
```

```
repeat
```

```
  <блок>
```

```
until X > 0
```

```
BEG:  
  <блок>  
CMP X, 0  
JG BEG  
...
```

Преодоление ограничения на длину безусловного перехода (для i8086)

>127 б

```
CMP X, Y  
JG XGY  
MOV AX, DX  
  ...  
XGY: ...
```

Relative jump out of range!

Команды

передачи
управления

```
CMP X, Y  
JLE XLEY  
JMP XGY  
XLEY: MOV AX, DX  
  ...  
XGY: ...
```

Счетные циклы

LOOP метка
(FOR I:=1 TO N do ...)

```
MOV  CX,N
CONT:
  <тело цикла>
DEC  CX    ;CX:=CX-1
CMP  CX,0
JNE  CONT
```

```
MOV  CX,N
CONT:
  <тело цикла>
LOOP CONT    ; CX:=CX-1
                ; IF CX#0 THEN GOTO CONT
```

- В качестве счетчика цикла используется только регистр CX;
- Тело цикла будет выполнено хотя бы один раз;
- Для i8086 LOOP использует только короткий переход;

```
;Пример:  Вычислить факториал N!
MOV  AX,1    ;начальное значение N!
MOV  CX,N    ;счетчик цикла = N
JCXZ  FIN    ;проверим CX (рекомендуется!)
MOV  SI,1    ;SI=1
F:   MUL  SI    ; (DX,AX:=AX*SI)
     INC  SI    ;SI:=SI+1
     LOOP F    ;повторять, пока CX#0
FIN:  ...
```

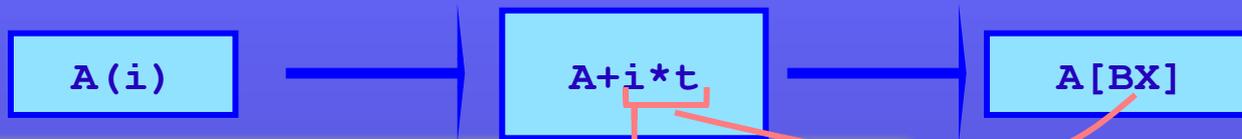
Счетные циклы с условием

LOOPE (LOOPZ) метка «Повторяй, пока 0»	LOOPNE (LOOPNZ) метка «Повторяй, пока не 0»
CX:=CX-1 IF (CX≠0) AND (ZF=1) THEN GOTO метка	CX:=CX-1 IF (CX≠0) AND (ZF=0) THEN GOTO метка

;ПРИМЕР Найти наименьшее число последовательности [2, K],
;на которое не делится число N (K, N – байтовые переменные)

```
MOV DL, N
MOV DH, 0          ; DX:=N
MOV CL, K
MOV CH, 0
DEC CX            ; CX:=K-1 (счетчик цикла)
MOV BL, 1
DV: INC BL        ; очередное число из диапазона [2, K]
MOV AX, DX
DIV BL            ; AH:=N mod BL (берем остаток)
CMP AH, 0        ; остаток=0?
LOOPE DV         ; цикл CX раз пока остаток=0
JNE DV1          ; остаток <> 0 → выход из программы
MOV BL, 0        ; нет искомого числа (запишем 0)
DV1: ...
```

Обработка одномерных массивов



```
MAS DB 1,0,-3,5,17
. . .
MOV AH,MAS+3 ;AH:=5 прямая адресация
MOV BX,3
MOV AL,MAS[BX] ;AL:=5 косвенная адресация
;Суммирование элементов массива байтов
```

```
MOV AL,0
MOV CX,5
MOV SI,0
NEXT: ADD AL,MAS[SI]
      ADD SI,1
      LOOP NEXT
```

MAS

1	0	-3	5	17
0	1	2	3	4

В качестве индексных регистров разрешается использовать BX,SI,DI (для i8086)

MAS2

5	0	8	0	1	0	FE	FF
0	1	2	3	4	5	6	7

```
MAS2 DW 5,8,1,-2
. . .
MOV AX,MAS2+4 ;AX:=1
MOV BX,4
MOV AX,MAS2[BX] ;AX:=1
;Суммирование элементов массива слов
MOV AX,0
MOV CX,4
MOV SI,0
NEXT: ADD AX,MAS2[SI]
      ADD SI,2
      LOOP NEXT
```

Обработка
массивов

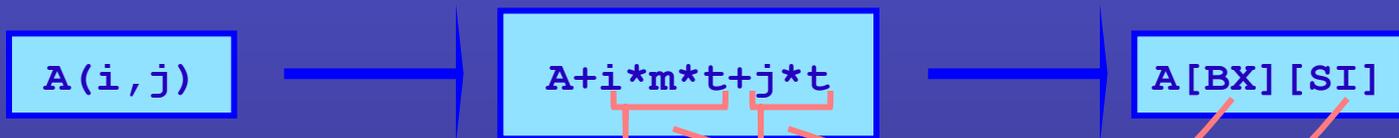
Обработка двумерных массивов

A DB 3,1,2
DB 5,4,6
DB 8,9,7

или

A DB 3,1,2,5,4,6,8,9,7

	A	A+1	A+2	A+3	A+4	A+5	A+6	A+7	A+8	
...	3	1	2	5	4	6	8	9	7	...
	i=0	i=0	i=0	i=1	i=1	i=1	i=2	i=2	i=2	
	j=0	j=1	j=2	j=0	j=1	j=2	j=0	j=1	j=2	
	BX=0			BX=3			BX=6			



A – начальный адрес массива;

m – количество элементов в строке; n – количество строк;

t – тип элемента массива (количество байт, занятых под один элемент).

$A[BX][SI] \Rightarrow A + (BX) + (SI)$ – адрес выбранного элемента

$BX := i * m * t$ – смещение первого элемента i-й строки

($i=0, 1, \dots, n$)

$SI := j * t$ – смещение j-го элемента в текущей строке ($j=0, 1, \dots, m$)

Обработка
МАССИВОВ

Пример работы с двумерными массивами

```
MAS3  DB  1,2,3,1
      DB  3,4,0,2
      DB  7,8,9,3
M     EQU 4
N     EQU 3
      . . .
;выбор элемента i=1,j=2
      MOV  BX,4           ;BX:=i*m*t=1*4*1=4
      MOV  SI,2          ;SI:=j*t=2*1=2
      MOV  AH,MAS3[BX][SI] ;AH:=0 (MAS3+4+2 - 6-й эл-т)
;выбор элемента i=i+1,j=1
      ADD  BX,M*TYPE MAS3 ;BX:=BX+m*t=BX+4=8
      MOV  SI,1          ;SI:=j*t=1*1=1
      MOV  AL,MAS3[BX][SI] ;AL:=8 (MAS3+8+1 - 9-й эл-т)
```

MAS3

1	2	3	1	3	4	0	2	7	8	9	3
0	1	2	3	4	5	6	7	8	9	10	11

Обработка

МАССИВОВ

Логические команды

- ❑ Команды логических операций
AND, OR, XOR, NOT, TEST
- ❑ Команды сдвига
SHR, SHL, SAR, SAL, ROR, ROL, ...
- ❑ Команды обработки бит
BSF, BSR, BT, BTC, BTR, BTS

Команды логических операций

Логическое умножение (И)

AND приемник, источник

TEST приемник, источник

Исключающее ИЛИ

XOR приемник, источник

Логическое сложение (ИЛИ)

OR приемник, источник

Логическое отрицание (НЕ)

NOT приемник

Приемник - m8, r8, m16, r16, m32, r32;

Источник -m8, r8, i8, m16, r16, i16, m32, r32, i32;

Формируют флаги ZF, SF, PF, флаги CF и OF сбрасываются в 0.

X	Y	X and Y	X or Y	X xor Y	not X
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

ЛОГИЧЕСКИ

в команды

Применение команд логических операций

Проверка значения бита числа

```
AND X,00000010B ;проверка 1-го бита
JZ  BIT_0        ;перейти, если бит=0
...
BIT_0:
```

AND	X ₇	X ₆	X ₅	X ₄	X ₃	X ₂	X ₁	X ₀
	0	0	0	0	0	0	1	0
<hr/>								
	0	0	0	0	0	0	X ₁	0

Установка значения бита числа

```
OR  X,00000010B ;установка 1-го бита в 1
AND X,11111101B ;сброс 1-го бита в 0
```

OR	X ₇	X ₆	X ₅	X ₄	X ₃	X ₂	X ₁	X ₀
	0	0	0	0	0	0	1	0
<hr/>								
	X ₇	X ₆	X ₅	X ₄	X ₃	X ₂	1	X ₀

Инвертирование значения бита

```
XOR X,00000010B ;инвертирование 1-го бита
```

AND	X ₇	X ₆	X ₅	X ₄	X ₃	X ₂	X ₁	X ₀
	1	1	1	1	1	1	0	1
<hr/>								
	X ₇	X ₆	X ₅	X ₄	X ₃	X ₂	0	X ₀

XOR	X ₇	X ₆	X ₅	X ₄	X ₃	X ₂	X ₁	X ₀
	0	0	0	0	0	0	1	0
<hr/>								
	X ₇	X ₆	X ₅	X ₄	X ₃	X ₂	X ₁	X ₀

Команды сдвига

Логический сдвиг

Арифметический сдвиг

Циклический сдвиг

SHR операнд, счетчик

SAR операнд, счетчик

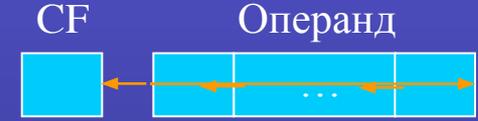
ROR операнд, счетчик



SHL операнд, счетчик

SAL операнд, счетчик

ROL операнд, счетчик



Операнд - m8, r8, m16, r16, m32, r32; счетчик - i8, CL;

```
MOV AL, 01000111B
SHL AL, 1
    ;AL=10001110, CF=0
SHL AL, 1
    ;AL=00011100, CF=1
```

```
MOV BH, 10001110B
SAR BH, 1
    ;BH=11000111, CF=0
MOV BH, 00001110B
SAR BH, 1
    ;BH=00000111, CF=0
```

```
MOV CL, 11000011B
ROL CL, 1
    ;CL=10000111, CF=1
MOV BL, 11100010B
ROR BL, 1
    ;BL=01110001, CF=0
```

Применение команд сдвига

Быстрое умножение на степени 2 ($X * 2^k$)

```
MOV AL, 5      ;AL=00000101b=5
SHL AL, 3      ;AL=00101000b=40=5*23,
SHL AL, 3      ;AL=01000000b=64≠40*23, результат неверный!
```

- Выполняется быстрее, чем умножение с помощью команды MUL
- Применяется для беззнаковых и знаковых (в дополнительном коде) чисел
- Дает верный результат, если старшая значащая цифра не выходит за пределы разрядной сетки

Быстрое целочисленное деление на степени 2 ($X / 2^k$)

```
MOV AL, 18     ;AL=00010010b=18
SHR AL, 3      ;AL=00000010b=2=18\23
MOV BL, -18    ;AL=11101110b
SAR BL, 2      ;AL=11111011b=-4=-18\22
```

- Выполняется быстрее, чем деление с помощью команды DIV
- Для беззнаковых чисел применяется команда SHR
- Для знаковых чисел (в дополнительном коде) применяется команда SAR (округляет частное в меньшую сторону!)

Цепочечные команды

Цепочка – последовательность элементов данных, записанных в памяти
(байтов, слов, двойных слов)

- Команды пересылки цепочек
MOVSB, MOVSW, MOVSD, MOVS
- Команды сравнения цепочек
CMPSB, CMPSW, CMPSD, CMPS
- Команды сканирования цепочек
SCASB, SCASW, SCASD, SCAS
- Команды извлечения элемента из цепочки
LODSB, LODSW, LODSD, LODS
- Команды заполнения цепочки
STOSB, STOSW, STOSD, STOS
- Префиксы повторения цепочечных команд
REP, REPE, REPZ, REPNE, REPNZ

Обобщенный формат цепочечных команд

[префикс_повторения] команда

✓ Неявные операнды цепочечных команд:

Приемник (результатирующая цепочка) – адресуется парой регистров ES : DI

Источник (исходная цепочка) – адресуется парой регистров DS : SI

✓ Направление (последовательность) обработки элементов цепочки:

Флаг DF = 0 - обработка вперед (от младших адресов к старшим) – по умолчанию

Флаг DF = 1 - обработка назад (от старших адресов к младшим)

CLD
(DF=0)

STD
(DF=1)

✓ Автоматическая модификация значений регистров SI и DI

	DF = 0	DF = 1
Байт	+1	-1
Слово	+2	-2
Дв. слово	+4	-4

Цепочечны
е команды

Обобщенный формат цепочечных команд

✓ Префикс повторения

REP – устанавливает повторение цепочечной команды N раз (задается в регистре CX);
REPE (REPNE) – повторение N раз, но пока ZF=0 (ZF≠0).

Префикс REP обеспечивает пересылку до 64 Кб для i8086 и до 4 Гб для i386

REP (CX:=CX-1, повтор, пока CX≠0)
REPE | REPZ (CX:=CX-1, повтор, пока ZF=1 и CX≠0)
REPNE | REPNZ (CX:=CX-1, повтор, пока ZF=0 и CX≠0)

```
L: if CX = 0 then goto L1
   CX:=CX-1
   <цепочечная команда>
   goto L
L1:
```

```
L: if CX = 0 then goto L1
   CX:=CX-1
   <цепочечная команда>
   if ZF=1 then goto L
L1:
```

```
L: if CX = 0 then goto L1
   CX:=CX-1
   <цепочечная команда>
   if ZF=0 then goto L
L1:
```

Порядок действий при использовании цепочечных команд

1. Установить значение флага DF в зависимости от направления обработки цепочек
2. Загрузить указатели на адреса цепочек в памяти в регистры DS:SI и ES:DI
3. Если количество обрабатываемых элементов больше 1, записать их число в регистр CX
4. Записать цепочечную команду с учетом типа элементов цепочки; если необходимо, использовать префикс повторения REP (REPE, REPNE).

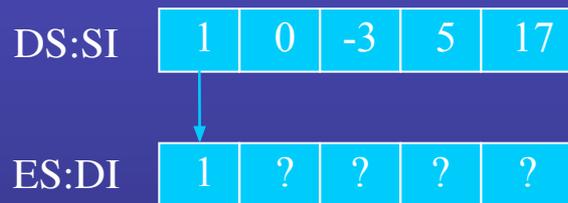
Команды пересылки цепочек

```
[REP] MOVSB ([ES:DI] := [DS:SI], DI := DI ± 1, SI ± 1)
[REP] MOVSW ([ES:DI] := [DS:SI], DI := DI ± 2, SI ± 2)
[REP] MOVSD ([ES:DI] := [DS:SI], DI := DI ± 4, SI ± 4)
```

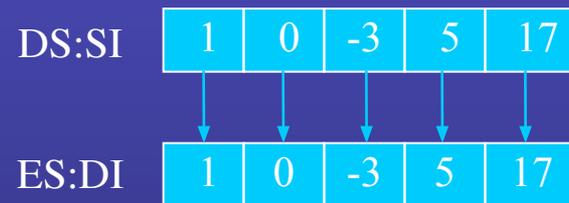
MOVSB — для цепочек байтов; MOVSW — для цепочек слов; MOVSD — для цепочек дв. слов
Операнды: DS:SI — адрес цепочки источника, ES:DI — адрес цепочки приемника.

MOVSB ; переслать 1 байт

MOV CX, 5
REP MOVSB ; переслать 5 байт



SI := SI + 1
DI := DI + 1

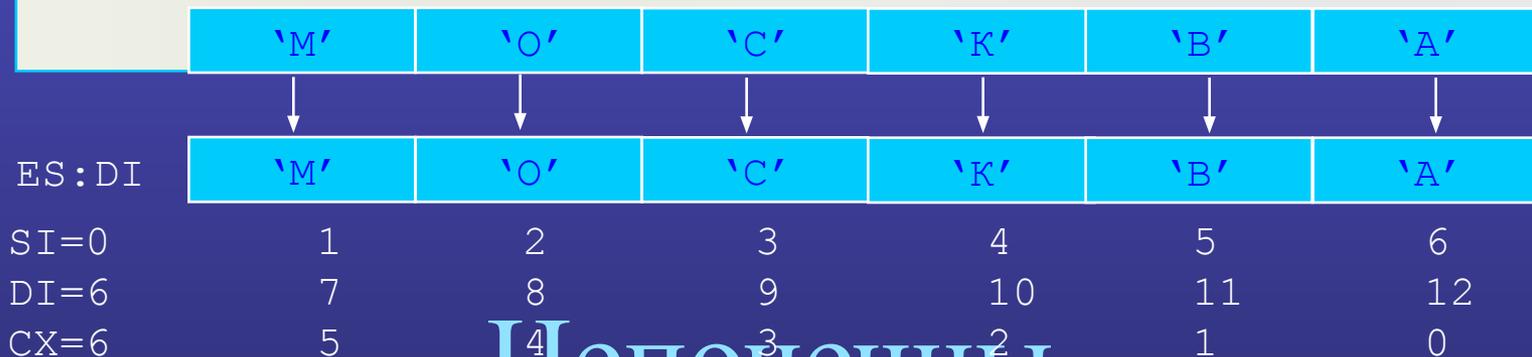


SI := SI + 5
DI := DI + 5
CX := 0

Цепочечны
е команды

Копирование блоков памяти с помощью MOVS

```
.DATA
STR1 DB 'МОСКВА' ; строка-источник
N EQU $-STR1 ; N - длина строки-источника
STR2 DB N DUP(?) ; строка-приемник
.CODE
S: MOV AX,@data
MOV DS,AX
MOV ES,AX ; загрузка ES (для адреса приемника)
CLD ; устанавливаем флаг DF=0
LEA SI,STR1 ; загрузка индексного регистра источника
LEA DI,STR2 ; загрузка индексного регистра приемника
MOV CX,N ; загрузка в CX числа элементов цепочки N
REP MOVSB ; пересылка N элементов байтовой цепочки
END S
```



Цепочны

в команды

Команды сравнение цепочек

```
[REP [N] E] CMPSB ([ES:DI] - [DS:SI] => флаг ZF DI:=DI±1, SI±1)
[REP [N] E] CMPSW ([ES:DI] - [DS:SI] => флаг ZF DI:=DI±2, SI±2)
[REP [N] E] CMPSD ([ES:DI] - [DS:SI] => флаг ZF DI:=DI±4, SI±4)
```

CMPSB – для цепочек байтов; CMPSW – для цепочек слов; CMPSD – для цепочек двойных слов
Операнды: DS:SI – адрес цепочки источника, ES:DI – адрес цепочки приемника

```
;Пример сравнения строк
CLD
MOV CX,5
LEA SI,STR1
LEA DI,STR2
REPE CMPSB
JE EQUAL
;строки различаются
...
EQUAL: ... ;есть совпадение
```

STR1	М	А	Р	И	Я
	=	=	=	≠	
STR2	М	А	Р	Т	А
	ZF=1	ZF=1	ZF=1	ZF=0	
	SI+1	SI+2	SI+3	SI+4	
	DI+1	DI+2	DI+3	DI+4	

SI, DI указывают на следующий элемент за элементом, давшим несовпадение

Цепочечны

е команды

Команды сканирования цепочек

```
[REP[N]E] SCASB ([ES:DI] - (AL) => флаг ZF, DI := DI ± 1)
[REP[N]E] SCASW ([ES:DI] - (AX) => флаг ZF, DI := DI ± 2)
[REP[N]E] SCASD ([ES:DI] - (EAX) => флаг ZF, DI := DI ± 4)
```

SCASB – для байтовых цепочек; SCASW – для цепочек слов; SCASD – для цепочек дв. слов
Операнды: ES : DI – адрес цепочки приемника, AL (AX, EAX) – искомый элемент

```
;Найти символ '.' в тексте и заменить его символом '!'.
STR DB 'Здравствуй, дорогая, и прощай.'
L EQU $-STR
...
MOV AL, '.'
LEA DI, STR ;загрузка DI
MOV CX, L ;счетчик повторений
CLD ;DF=0
REPNE SCASB ;сравнивать эл-ты строки с содержимым AL до
;тех пор, пока не будет обнаружен первый эл-т,
;совпадающий с символом '.'
JNE FIN ;если не найден д - перейти к FIN
MOV BYTE PTR ES:[DI-1], '!' ; поместить новый символ
FIN: ...
```

Команды заполнения цепочки

```
[REP] STOSB ([ES:DI] := (AL) ,DI:=DI±1)
[REP] STOSW ([ES:DI] := (AX) ,DI:=DI±2)
[REP] STOSD ([ES:DI] := (EAX) ,DI:=DI±4)
```

STOSB – для байтовой цепочки; STOSW – для цепочки слов; STOSD – для цепочки дв. слов.

Операнды: ES:DI – адрес цепочки приемника, AL (AX, EAX) – элемент-источник

```
;Заполнить область памяти символами '*'
STR  DB 32 DUP(?)
...
LEA  DI,STR    ;загрузка DI
MOV  CX,32     ;счетчик повторений
CLD                          ;DF=0
MOV  AL,'*'
REP  STOSB     ;область памяти заполняется символами '*'
```

Команды извлечение элементов из цепочки

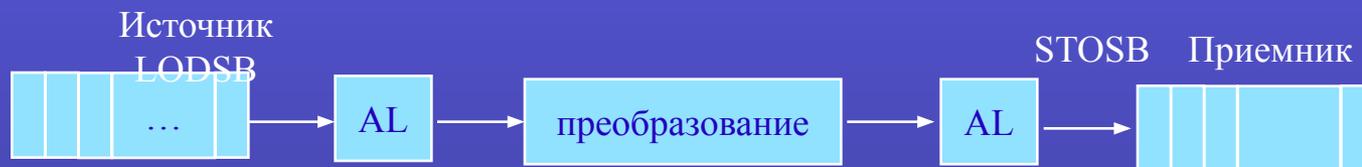
LODSB (**AL** := (**DS:SI**) , **SI** := **SI±1**)

LODSW (**AX** := (**DS:SI**) , **SI** := **SI±2**)

LODSD (**EAX** := (**DS:SI**) , **SI** := **SI±4**)

LODSB – для байтовой цепочки; LODSW – для цепочки слов; LODSD – для цепочки дв. слов

Операнды: DS:SI – адрес цепочки источника, AL (AX, EAX) – элемент-приемник



;Переписать эл-ты байтового массива X в байтовый массив Y
;с инверсией знака (число элементов 100)

```
CLD          ;DF=0
LEA SI,X    ;загрузка SI (для команды LODS)
LEA DI,Y    ;загрузка DI (для команды STOS)
MOV CX,100  ;счетчик повторений
L: LODSB    ;Xi -> AL, SI:=SI+1
NEG AL      ;изменение знака
STOSB      ;AL -> Yi, DI:=DI+1
LOOP L      ;повторить 100 раз
```

Структуры

- ❑ Описание шаблона структуры (список полей данных)
STRUC, ENDS
- ❑ Определение экземпляров структуры в сегменте данных (выделение памяти и инициализация)
- ❑ Организация доступа к элементам (полям) структуры. Оператор `.`

Описание структуры (шаблон)

```
имя STRUC  
<описание поля1> (директива DB|DW|DD)  
.  
.  
.  
<описание поляN> (директива DB|DW|DD)  
имя ENDS
```

```
DATE STRUC ;дата  
    Y DW 1994 ;год  
    M DB 3 ;месяц  
    D DB ? ;день  
DATE ENDS
```

```
STUD STRUC ;студент  
FAM DB 12 DUP ( ' ' );фамилия  
NANE DB ' ' ;имя  
SEX DB 'M' ;пол  
BYEAR DW ? ;год рожд.  
MARKS DB 4 DUP (?) ;оценки  
STUD ENDS
```

Определение экземпляров структур

`имя_переменной имя_структуры [<список значений>]`

```
D1  DATE  <?, 6, 9>
D2  DATE  <1998,,>
D3  DATE  <,,>
```

D1

D2

D3

?	6	9	1998	3	?	1994	3	?
---	---	---	------	---	---	------	---	---

TYPE DATE=4 (по количеству байт, зарезервированных в структуре)
TYPE D1=4, TYPE STUD=28

Массивы структур

(таблицы)

```
DATES    DATE    100 DUP < > ; (100 дат)
UAI_211  STUD    25  DUP < >
UIS_211  STUD    'Иванов', 'Иван', 'М', 1983, 3, 3, 3, 3
          STUD    'Петров', 'Петр', ' М', 1982, 4, 3, 2, 5
          STUD    'Сидорова', 'Анна', 'Ж', 1982, 5, 4, 5, 4
L EQU TYPE UIS_211 ; длина строки таблицы
```

Структуры

данные

Доступ к элементам структур

```
имя_структуры[±С].имя_поля  
[регистр][±С].имя_поля
```

Оператор (.) относится к адресным выражениям и обозначает адрес, вычисляемый по формуле: (<адресное выражение>+<смещение поля в структуре>)

Тип адреса совпадает с типом (размером) указанного поля.

Прямая адресация

```
D1.Y ; (D1 + 0)  
D1.D ; (D1 + 3)
```

Косвенная адресация

```
LEA BX, D2  
MOV AX, [BX].Y ;AX:=1998  
K EQU TYPE D2.Y ;K=2 байт
```

Использование структур: пример 1

```
;Подсчитать количество студентов мужского пола
.MODEL SMALL
STUD  STRUC
. . .
STUD  ENDS
.DATA
GR    STUD  'Иванов' , 'Иван' , 'М' , 1983, 3, 3, 3, 3
      STUD  'Сидоров' , 'Петр' , 'М' , 1980, 4, 4, 3, 2
      STUD  'Петрова' , 'Маша' , 'Ж' , 1984, 5, 4, 4, 3
. . .
LEN   EQU   TYPE STUD           ;длина строки таблицы
N     EQU   ($-GR)/LEN         ;количество строк в таблице
.CODE
. . .
      MOV   AL, 0                ;счетчик лиц 'М'
      MOV   CX, N                ;счетчик строк в таблице
      MOV   BX, 0                ;смещение текущей строки таблицы
CYCLE CMP   (GR[BX]).SEX, 'М'    ;сравнить поле SEX текущей строки с 'М'
      JNE   NEXT                ;если не равно - дальше
      INC   AL                  ;если равно - AL:=AL+1
NEXT:  ADD   BX, LEN             ;указатель на следующую строку
      LOOP CYCLE                ;продолжить, если CX<>0
```

Использование структур: пример 2

;Посчитать количество студентов с именем Иван

. . .

```
NAMEF DB 'Иван'
N1 EQU $-NAMEF ;длина NAMEF
.CODE
    MOV AX,@data
    MOV DS,AX
    MOV ES,AX ;регистр ES нужен для цепочечной команды
    CLD ;направление обработки цепочки (DF=0)
    MOV AL,0 ;счетчик совпадений с именем 'Иван'
    MOV CX,N ;счетчик цикла (N-число студентов)
    LEA BX,GR.NAME ;адрес поля NAME в BX
CYCLE: LEA DI,NAMEF ;индекс цепочки-источника (строка NAMEF)
    MOV SI,BX ;индекс цепочки-приемника (поле GR.NAME)
    MOV DX,CX ;спасаем CX
    MOV CX,N1 ;в CX - длина цепочки (4)
    REPE CMPSB ;сравнивать цепочки, пока равно
    JNE NEXT ;если не совпали - дальше
    INC AL ;если совпали -счетчик совпадений AL:=AL+1
NEXT: ADD BX,LEN ;поместить в BX адрес поля NAME след.строки
    MOV CX,DX ;восстанавливаем счетчик цикла
    LOOP CYCLE ;продолжить, если CX<>0
```

Макросредства

- ❑ Макроопределения (описание макроса)

MACRO, ENDM

- ❑ Макрокоманды (вызов макроса)

- ❑ Макрогенерация и макроподстановка

- ❑ Макродирективы

WHILE, REPT, IRP, EXITM, GOTO, IF

Структуры

данных

Макроопределение

```
имя_макроста MACRO [список_формальных_параметров]  
<тело макроста>  
ENDM
```

Размещение макроопределения в программе

- в начале программы (до сегмента данных);
- в отдельном файле (include имя файла)

```
;Настройка DS на сегмент данных  
SETDS MACRO  
    MOV    AX,@data  
    MOV    DS,AX  
ENDM
```

```
;Вывод строки символов ASCII  
OUTSTR MACRO STR  
    MOV    AH,9  
    MOV    DX,OFFSET STR  
    INT    21h  
ENDM
```

```
;Суммирование слов X:=X+Y  
SUM MACRO X,Y  
    MOV    AX,Y  
    ADD    X,AX  
ENDM
```

Вызов макросов: макрокоманды

имя_макроса [список_фактических_параметров]

```
INCLUDE C:\TASM\MACRO.ASM
.DATA
S DB 'Hello, world!$'
M1 DW 7
M2 DW 15
.CODE
START:
  SETDS
  OUTSTR S
  SUM M1,M2
  STOP 0
END START
STOP MACRO RC
  MOV AH,4Ch
  MOV AL,RC
  INT 21h
ENDM
```

```
.DATA
S DB 'Hello, world!$'
M1 DW 7
M2 DW 15
.CODE
START:
  MOV AX,@data
  MOV DS,AX
  MOV AH,9
  MOV DX,OFFSET S
  INT 21h
  MOV AX,M2
  ADD M1,AX
  MOV AH,4Ch
  MOV AL,0
  INT 21h
END START
```

макрос

Ассемблирование программ, содержащих макросы

Исходная программа

```
INCLUDE . . .  
. . .  
SETDS  
. . .  
STOP 3  
. . .  
STOP MACRO RC  
. . .  
ENDM
```

Макрогенерация

```
. . .  
MOV AX, @data  
MOV DS, AX  
. . .  
MOV AH, 4Ch  
MOV AL, 3  
INT 21h  
. . .
```

Ассемблирование

Объектный код

```
58E3F2007D2A11  
CD094687299FD9  
50AC2300FC472F  
1A85B205810EA6  
7F3BC58E3F2007  
D2A11CD0946872  
99FD950AC2300F  
C472F1A85B2058  
10EA6D950AC2C  
23
```

- **Макрогенерация:** поиск макрокоманд в исходной программе и замена на последовательность команд, описанных в соответствующих макроопределениях, с заменой формальных параметров на фактические (макроподстановка)
- **Ассемблирование:** создание объектного кода из полученного исходного текста.

Макросред

ства

96

Особенности использования макросов

- ❑ Программист обязан следить за соответствием типов формальных и фактических параметров и возможностью использования тех или иных типов в конкретных командах тела макроса.
- ❑ Фактические параметры, содержащие пробелы или разделители («,» или «.»), должны быть заключены в угловые скобки <>, например <BYTE PTR [BX]>
- ❑ В качестве формальных параметров можно использовать не только операнды команд, но и любые лексемы и их последовательности в теле макроса
- ❑ При использовании меток в теле макроса следует описать их с помощью директивы LOCAL список_имен (для автоматической замены повторяющихся меток)
- ❑ Если в теле макроса используются регистры процессора, то нужно иметь в виду, что содержимое этих регистров возможно будет испорчено.

Макродирективы

Макродирективы повторения:

WHILE

REPT

IRP

Директивы управляются процессом
макрогенерации:

EXITM

GOTO

Макродиректива условной компиляции:

IF

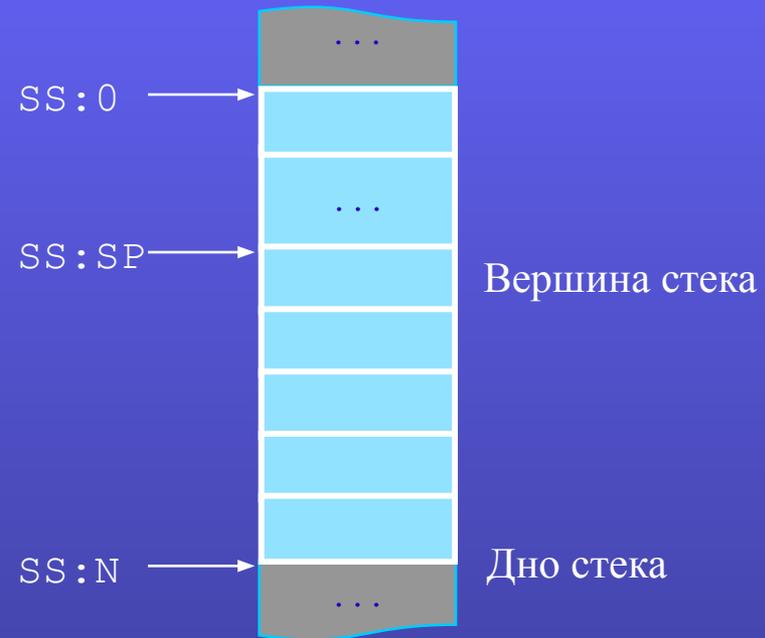
Макросред
ства

Процедуры в ассемблере

- ❑ Стек и стековые команды
PUSH, POP, PUSHF, POPF, PUSHA, POPA
- ❑ Описание процедуры
PROC, ENDP
- ❑ Вызов процедуры и возврат
CALL, RET
- ❑ Интерфейс с процедурой (передача параметров)
 - через регистры
 - через память
 - через стек
- ❑ Сравнительный анализ макросов и процедур
- ❑ Модульное программирование
- ❑ Прерывания и системные вызовы
INT, IRET

Стек

```
.MODEL SMALL  
.STACK N  
.DATA  
. . .  
.CODE  
. . .
```



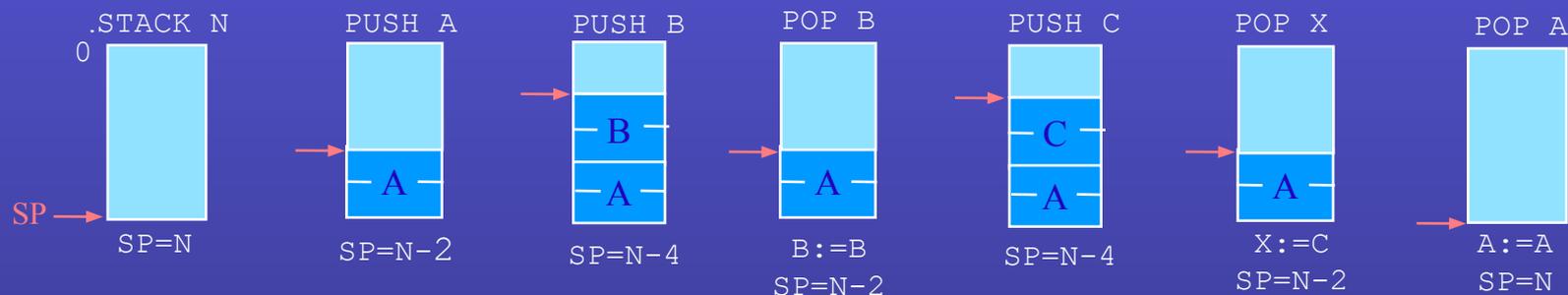
- N – размер стека (количество байтов в сегменте стека) ($\leq 64\text{К}$ для i8086);
- SS – сегментный регистр стека;
- SP – указатель вершины стека (начальное значение $SP=N$);
- SS:N – дно стека (положение неизменно);
- SS:SP – вершина стека (положение меняется);
- BP – альтернативный указатель стека.

Работа со стеком

PUSH операнд ($SP := SP - 2, [SS:SP] := \text{операнд}$)
POP операнд ($\text{операнд} := [SS:SP], SP := SP + 2$)

Тип операнда - r16, sr16, m16 (i16 для i80186); недопустимая команда POP CS

PUSHF ($SP := SP - 2, [SS:SP] := \text{FLAGS}$)
POPF ($\text{FLAGS} := [SS:SP], SP := SP + 2$)



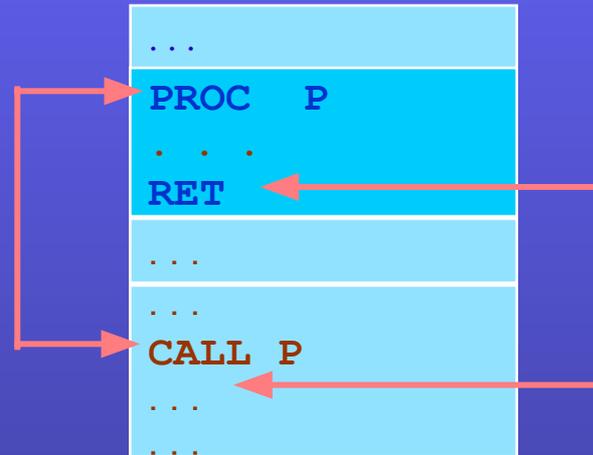
Назначение стека

- Временное сохранение данных (регистров);
- Обмен данными между регистрами или ячейками памяти:

```
PUSH AX  
POP BX ; BX:=AX
```

- Реализация механизма вызова процедур;
- Передача параметров при вызове процедур;

Процедуры (безусловный переход с возвратом)



Использование процедур:

- Описание процедуры (директива PROC);
- Вызов процедуры (команда CALL);
- Возврат из процедуры в точку вызова (команда RET);
- Передача параметров при вызове процедур;

Описание процедуры

```
имя_процедуры PROC [NEAR|FAR]  
<тело процедуры>      (последовательность команд)  
[имя_процедуры] ENDP
```

- Имя процедуры обрабатывается ассемблером как метка;
- Тело процедуры состоит из последовательности произвольных команд;
- NEAR – процедура доступна только из того сегмента, где она описана (по умолчанию);
- FAR – процедура может быть вызвана из других сегментов;

Процедура может быть описана:

- Между директивой `.code` и точкой входа (меткой `start`);
- После последней команды сегмента кода;
- В отдельном сегменте кода;
- В другом модуле (исходном файле).

Вызов процедуры

```
CALL [SHORT] имя
(SP:=SP-2, [SS:SP]:=IP, IP:=offset имя)
CALL FAR PTR имя
(SP:=SP-2, [SS:SP]:=CS, SP:=SP-2, [SS:SP]:=IP,
CS:=seg имя, IP:=offset имя)
```

- Вызов процедуры выполняется как безусловный переход:
JMP имя_процедуры
- Модификатор SHORT (короткий переход) требует формировать однобайтное смещение для адреса (-128 .. +127)
- В стеке сохраняется адрес возврата из процедуры (содержится в регистрах CS : IP перед выполнением команды CALL)
 - NEAR вызов (внутрисегментный, ближний) сохраняет только IP (EIP)
 - FAR вызов (межсегментный, дальний) сохраняет и CS и IP (EIP)
- Допускаются вложенные вызовы процедур

Возврат из процедуры

RET [число]

Ближний: $IP := [SS:SP]$, $SP := SP + 2 + \text{число}$

Дальний: $IP := [SS:SP]$, $CS := [SS:SP+2]$, $SP := SP + 4 + \text{число}$

- Извлекает из стека ранее сохраненный адрес возврата:
 - IP для ближнего вызова
 - CS и IP для дальнего вызова
- Тип возврата ассемблер определяет автоматически (желательно описание процедуры производить ранее, чем был указан ее вызов!);
- Удаляет из стека заданное число (тип $i16$) байтов ($SP := SP + \text{число}$) (без учета адреса возврата!);
- Передает управление по адресу $CS:IP$

Интерфейс с процедурой: методы передачи параметров

Что передавать?

- передача по значению – передается значение параметра:

procedure P (X : integer)

- передача по ссылке – передается адрес параметра:

procedure P (var X : integer)

Как передавать?

- через регистры;
- через стек;
- через общие области памяти;

Передача параметров по значению с использованием регистров

Пример: Вычислить
 $R = \max(a, b) + \max(c, d)$

```
; Процедура AX:=max(AH, BH)
; Параметры: AX-первое число, BX-второе число
; Результат → в AX
```

```
MAX PROC
    CMP AX, BX        ; сравниваем (AX) и (BX)
    JGE MX           ; если (AX) больше - выход
    MOV AX, BX       ; иначе AX:=BX
MX: RET
ENDP
```

```
; Основная программа
```

```
. . .
MOV AX, A           ; подготовка параметров в AX и BX
MOV BX, B           ; для вызова процедуры
CALL MAX            ; AX:=MAX(A, B)
PUSH AX             ; спасти AX
MOV AX, C           ; подготовка параметров для
MOV BX, D           ; нового вызова процедуры
CALL MAX            ; AX:=max(C, D)
POP R               ; R:=MAX(A, B)
ADD R, AX           ; R:=R+MAX(C, D)
. . .
```

Передача параметров по ссылке с использованием регистров

```
;Процедура AX:= $\Sigma(X_i)$ , i=1..N  
;Параметры: BX - адрес массива, CX - число эл-тов  
;Результат (сумма элементов) → в AX
```

```
SUM PROC  
    PUSH SI                ;спасаем SI  
    MOV AX,0               ;обнулить сумму  
    MOV SI,0               ;индекс массива (SI)=0  
NXT: ADD AX,[BX][SI]       ;AX:=AX+Xi  
    ADD SI,2               ;SI:=SI+2 только для слов!  
    LOOP NXT  
    POP SI  
    RET  
ENDP
```

```
;Основная программа
```

```
. . .  
X    DW 1,2,3,4,5  
MAS  DW 7,8,9  
  
. . .  
    LEA BX,X               ;адрес начала массива в BX  
    MOV CX,5               ;число элементов массива в CX  
    CALL SUM  
  
. . .  
    LEA BX,MAS             ;адрес начала массива в BX  
    MOV CX,3               ;число элементов массива в CX  
    CALL SUM
```

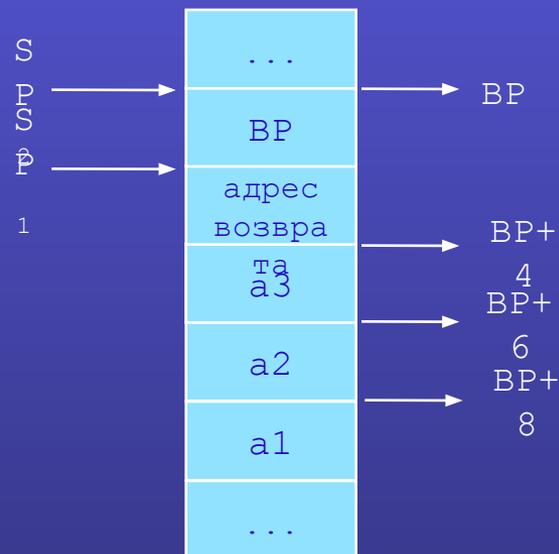
Передача параметров через стек

Procedure p (a1,a2,a3)

```
;Вызов процедуры P
PUSH A1
PUSH A2
PUSH A3
CALL P
```

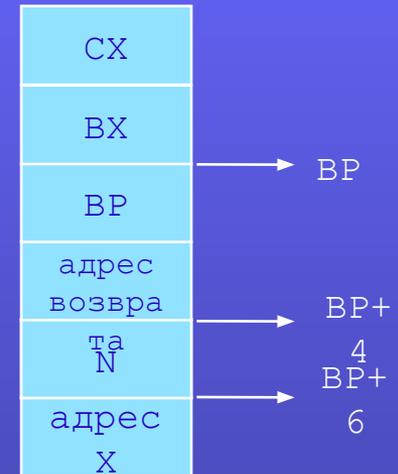
- Вызывающая программа:
 - размещает параметры в стеке
 - осуществляет вызов процедуры
- Процедура:
 - в начале работы извлекает параметры из стека
 - перед завершением работы очищает стек.

```
;Процедура P
P PROC NEAR
PUSH BP
MOV BP,SP ; BP:=SP
. . .
MOV AX,[BP+4] ; AX:=a3
MOV BX,[BP+6] ; BX:=a2
MOV CX,[BP+8] ; CX:=a1
. . .
POP BP ;убрать BP из стека
RET 6 ;убрать параметры
END P
```



Пример (суммирование элементов массива)

```
; Процедура AX:=Σ(Xi), i=1..N
SUM  PROC NEAR
      PUSH BP          ; пролог процедуры
      MOV BP,SP        ;
      PUSH BX          ; спасаем регистр BX
      PUSH CX          ; спасаем регистр CX
      MOV CX,[BP+4]    ; 2-й параметр - N
      MOV BX,[BP+6]    ; 1-й параметр - адрес массива
      MOV AX,0         ; обнулить сумму
NEXT: ADD AX,[BX]      ; AX:=AX+Xi
      ADD BX,2         ; BX:=BX+2 (для слов!)
      LOOP NEXT
      POP CX           ; восстанавливаем регистры
      POP BX
      POP BP
      RET 4
      ENDP
```



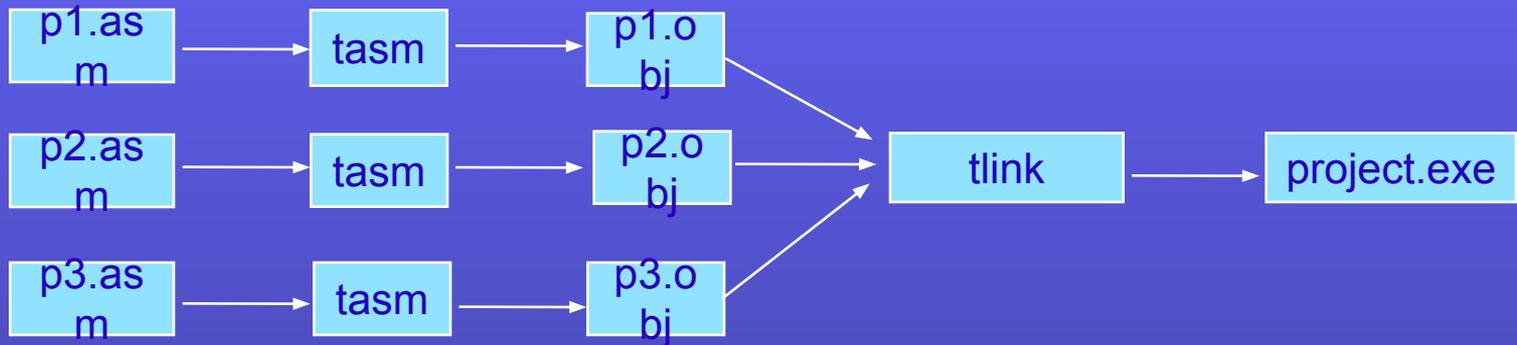
```
;Основная программа
X  DW 1,2,3,8,12
N  EQU 5
...
LEA AX,X      ;адрес массива
PUSH AX       ;первый параметр → в стек
MOV AX,N      ;число элементов массива
PUSH AX       ;второй параметр → в стек
CALL SUM      ;вызов процедуры
...
```

Сравнительный анализ макросов и процедур

Макросы	Процедуры
Многократное использование в программе единожды описанного кода	
Текст изменяется в зависимости от параметров	Текст неизменен
Текст добавляется к программе при каждом вызове	Текст в одном экземпляре, при вызове передается управление
Более эффективный по скорости код	Уступают в быстродействии
Не эффективно использует память	Более эффективны с точки зрения использования памяти

```
;Пример (комбинирование макросов и процедур)
SUMMA MACRO MAS,N
    LEA AX,MAS
    PUSH AX
    MOV AX,N
    PUSH AX
    CALL SUM
SUMMA ENDM
. . .
SUMMA X,5
```

Модульное программирование на Ассемблере



TLINK P1.OBJ P2.OBJ P3.OBJ PROJECT.EXE

Средства разрешения внешних ссылок

EXTRN имя:тип [{ ,имя:тип }]

Объекты, которые используются в этом модуле, но декларируются вне его
Имя: переменная (BYTE, WORD, ...), процедура или метка (NEAR, FAR), константа (ABC)

PUBLIC имя [{ ,имя }]

Объекты, которые декларированы в данном модуле и должны быть доступны в других модулях.

Пример многомодульной программы

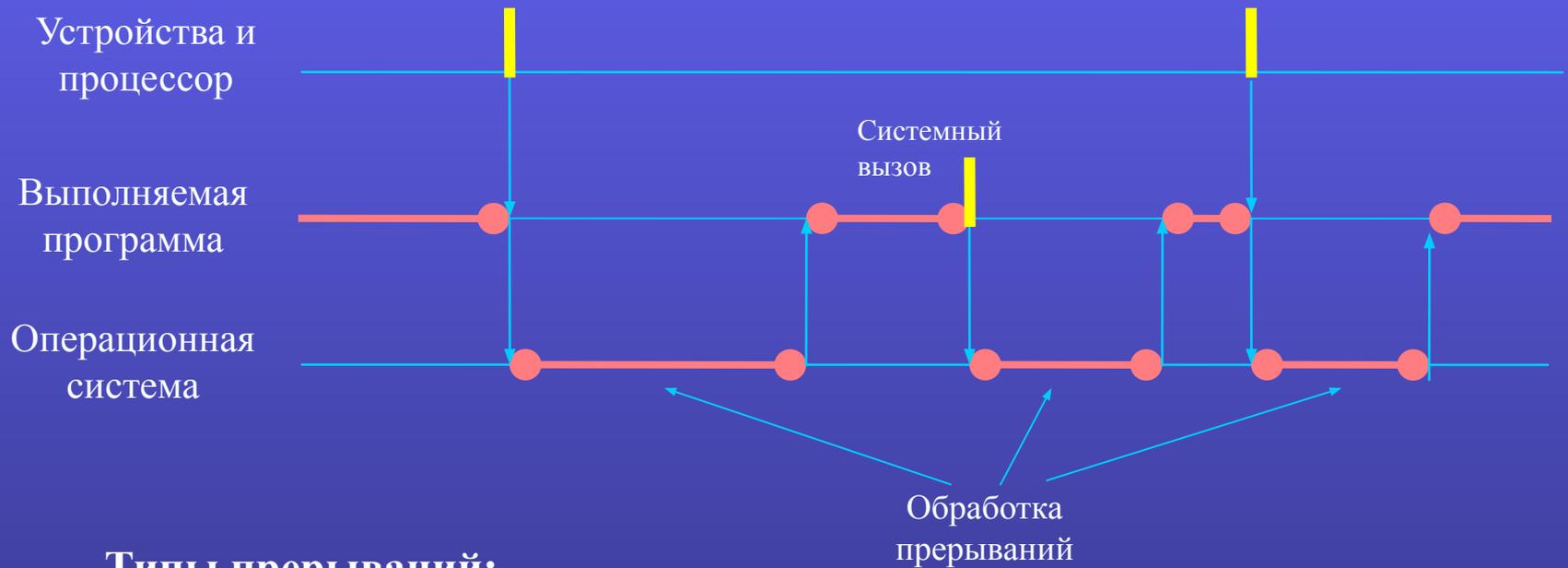
```
;МОДУЛЬ 1 (MOD1.ASM)
.MODEL SMALL
.DATA
. . .
.CODE
PROC1 PROC FAR
. . .
PROC1 ENDP
PUBLIC PROC1 ;Общее имя
                ;для всех
. . .
END ;Метка не указывается!
```

```
;МОДУЛЬ 2 (MOD2.ASM)
.MODEL SMALL
.STACK 256
.DATA
. . .
.CODE
EXTRN PROC1:FAR ;Внешнее имя
START:
. . .
CALL PROC1
. . .
END START ;Общая точка входа
```

Порядок создания многомодульной программы:

```
TASM MOD1.ASM
TASM MOD2.ASM
TLINK MOD1.OBJ MOD2.OBJ PRG.EXE
```

Прерывания и системные вызовы

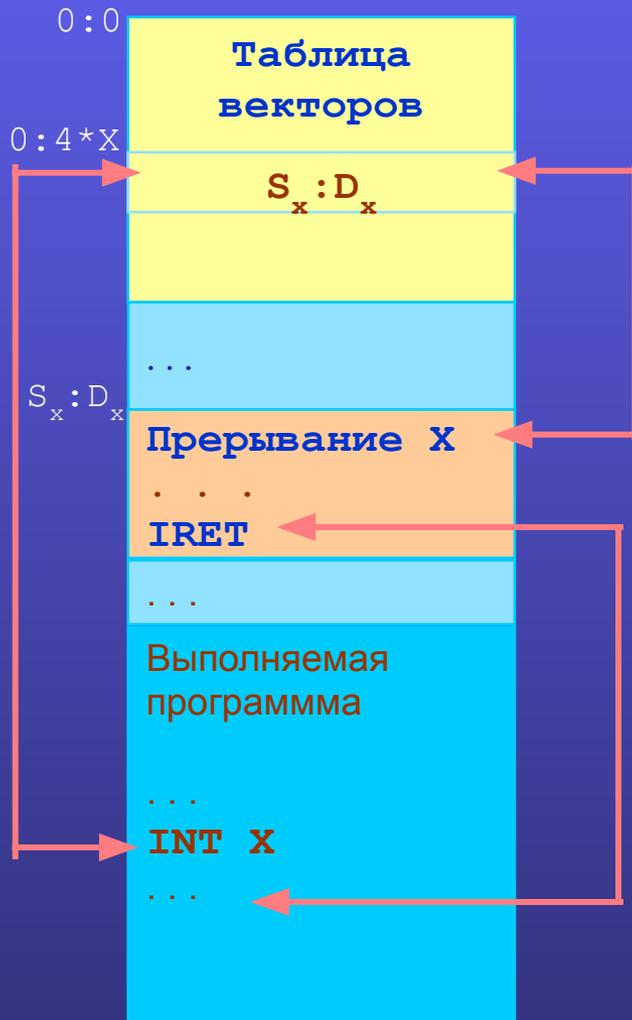


Типы прерываний:

- аппаратные (внешние)
- программные (системный вызов)
- исключения

Механизм обработки прерываний

Прерывание = процедура обработки прерывания



- ❑ Прерывания входят в состав ОС и различаются по номерам ($X=0\div 255$);
- ❑ Адреса всех прерываний хранятся в системной таблице векторов прерываний
- ❑ Вызов прерывания (команда `INT`): для внешних прерываний и исключений – автоматически, для системных вызовов – явно;
- ❑ Возврат из прерывания в точку вызова (команда `IRET`);
- ❑ Передача параметров при вызове прерываний осуществляется через регистры;

Вызов и возврат из прерывания

INT номер

$(SP := SP - 2, [SS:SP] := \text{Flags}, SP := SP - 2, [SS:SP] := CS, SP := SP - 2, [SS:SP] := IP, CS:IP := [0:4 * \text{номер}])$

- запись в стек регистра флагов (Flags) и адреса возврата (CS:IP);
- дальний переход (JMP FAR) по адресу, извлекаемому из таблицы векторов прерываний в соответствии с номером прерывания (по смещению $4 * \text{номер}$);

IRET

$IP := [SS:SP], CS := [SS:SP + 2], \text{Flags} := [SS:SP + 4], SP := SP + 6$

- извлечение из стека ранее сохраненных регистров (CS:IP и Flags)
- передача управления по адресу CS:IP (возобновление выполнения прерванной программы)



Использование прерываний

- Размещение прерываний
 - 0÷31 – поддерживаются аппаратно (входят в состав BIOS)
 - 32÷255 – поддерживаются программно (входят в состав ОС)
- Назначение прерываний
 - 0,1,3,4 – обработка исключений
 - 2,8÷15, – обработка аппаратных прерываний
 - остальные – обработка системных вызовов (программные прерывания, которые можно вызывать командой INT)
- Некоторые прерывания (16, 20, 23, 33 и др.) включают несколько различных функций. При вызове функций следует дополнительно указывать номер функции, записывая его в регистр AH
- Перед вызовом программного прерывания, следует записать в указанные регистры входные параметры (если потребуется)
- Результаты работы программного прерывания возвращаются также через регистры

```
; Вывод символа на экран
MOV AH, 9
MOV AL, код_символа
MOV BL, атрибут_символа
MOV CX, число_повторений
INT 10h
```

```
; Чтение символа с экрана
MOV AH, 8
INT 10h
;AL=код_символа
;AH=атрибут_символа
```

Примеры использования прерывания 33 (21h) MS DOS

```
; Завершение программы
MOV AH,4Ch
MOV AL,код_завершения
INT 21h
```

```
; Вывод символа на экран
MOV AH,2
MOV DL,код символа
INT 21h
```

```
; Вывод строки на экран
.data
STR DB 'Hello!$'
...
MOV AH,9
LEA DX,STR
INT 21h
```

```
; Удаление файла
.data
FName DB 'C:\COMMAND.COM'
MES1 DB 'Файл не найден$'
MES2 DB 'Файл удален!$'
...
MOV AH,41h
LEA DX,FName
INT 21h
JNC succ ;CF=0 -> успех
MOV AH,9 ;CF=1 -> ошибка
LEA DX,MES1
INT 21h
JMP fin
succ:
MOV AH,9
LEA DX,MES2
INT 21h
fin: . . .
```

Сегментирование памяти

- ❑ *Сегмент памяти* – непрерывная область памяти, предназначенная для размещения каких-либо блоков программы (данных, кода)
- ❑ Размер сегмента
 - для шестнадцатиразрядных приложений не более $2^{16} = 64$ Кб;
 - для тридцатидвухразрядных приложений не более $2^{32} = 4$ Гб.
- ❑ Сегменты могут размещаться в памяти произвольно по отношению друг к другу (даже пересекаться или совпадать)
- ❑ В командах обычно указывают эффективный адрес (смещение) операнда, а сегментная часть адреса извлекается по умолчанию из одного из 6 сегментных регистров (CS, DS, SS, ES, GS, FS).
- ❑ Сегментные регистры используются по умолчанию для формирования адреса в командах:
 - DS – команды пересылки, арифметические, логические и др., использующие данные в памяти;
 - CS – команды перехода (JMP, CALL);
 - SS – команды работы со стеком (PUSH, POP) и команды, использующие косвенную адресацию по BP;Остальные сегментные регистры никакой привязки изначально не имеют.
- ❑ Можно использовать адресацию с явным определением сегментного регистра:

```
MOV AX, ES:MEM
JMP GS:[BX]
```

Программ

ные

119

Определение программных сегментов

```
имя_сегмента SEGMENT [выравнивание] [объединение] [класс] [размер]  
<предложения Ассемблера>  
имя_сегмента ENDS
```

выравнивание

BYTE | WORD | DWORD | PARA – устанавливает, что адрес начала сегмента должен быть кратен соответственно байту, слову, двойному слову, параграфу (16б)

объединение

показывает, как данный сегмент будет объединяться с одноименными сегментами
PRIVATE – не объединять с другими сегментами (используется по умолчанию);

PUBLIC – будет выделен один общий программный сегмент для всех одноименных сегментов;

COMMON – будет выделен один сегмент памяти для всех одноименных сегментов (наложение);

STACK – определение сегмента стека (объединяются все сегменты)

класс

символическое имя в кавычках, которое используется как дополнительный признак для размещения или объединения сегмента

USE16 | USE32 – использовать 16-ти или 32-х разрядное смещение (IP/EIP)

Программ

ные

120

Программные сегменты

```
;-----  
A  SEGMENT  
A1  DB  400  DUP(?)  
A2  DW  8  
A  ENDS  
  
;-----  
B  SEGMENT  
B1  DW  A2  
B2  DD  A2  
B  ENDS  
  
;-----  
C  SEGMENT  
L:  MOV  BX,B1  
.  
.  
.  
C  ENDS  
  
;-----
```

- Все сегменты равноправны (не несут информации о их содержимом)
- Сегменты кода следует размещать после сегментов данных!!

Директива ASSUME

```
ASSUME {sr:имя_сегмента[,]}
```

- ❑ Устанавливает соответствие между программными сегментами и сегментными регистрами (sr)
- ❑ Загрузка регистров не производится!

```
;Пример 1 (см. стр. 113)  
ASSUME ES:A, DS:B, CS:C  
MOV AX, A2      ;AX:=(ES:A2)  
MOV BX, B1      ;BX:=(DS:B1)
```

ES и DS программист должен формировать сам; SS – формируется автоматически, если сегмент описан с ключевым словом STACK; CS – формируется автоматически по адресу стартовой метки кода, установленной в END или по значению, указанному в ASSUME.

```
;Пример 2  
C1 SEGMENT  
ASSUME CS:C1  
. . .  
JMP FAR PTR L  
. . .  
C1 ENDS  
C2 SEGMENT  
ASSUME CS:C1  
. . .  
L:  
. . .  
C2 ENDS
```

Программ

ные

122

Модели памяти

`.MODEL [модификатор] модель`

Модель:

- TINY – код, данные, и стек будут размещаться в одном сегменте памяти размером 64 Кб (используется при создании программ формата .com)
- SMALL – один сегмент для кода, один сегмент для.
- MEDIUM – несколько сегментов кода, один сегмент данных.
- COMPACT – один сегмент кода и несколько сегментов данных.
- LARGE – несколько сегментов кода и несколько сегментов данных.
- FLAT – один сегмент на все. Совпадает с TINY, но занимает область до 4 Гб.

Модификатор

`USE16 | USE32`

Программ
ные

123

Упрощенные директивы определения сегментов

```
.CODE [имя]  
_TEXT SEGMENT WORD PUBLIC 'CODE'
```

- Это определение действует для моделей TINY, SMALL, COMPACT
- Устанавливает CS на начало сегмента кода (ASSUME CS:_TEXT)

```
.STACK [размер]  
STACK SEGMENT PARA PUBLIC 'STACK'
```

```
.DATA  
_DATA SEGMENT WORD PUBLIC 'DATA'
```

Константы, действующие при использовании директивы .MODEL

@code – принимает значение адреса сегмента кода;

@data – принимает значение адреса сегмента данных;

@stack – принимает значение адреса сегмента стека.

Программ

Порядок описания и загрузки сегментов

- Сегменты описывают в программе в любом порядке.
- Сегмент кода следует размещать ниже сегмента данных, чтобы исключить проблему неопределенности адресов переменных;
- Сегменты одного класса размещаются рядом;
- Все сегменты типа PUBLIC объединяются в одном сегмент памяти, если они имеют одинаковые имена;
- Сегменты с общей директивой COMMON накладываются в памяти друг на друга (в одном сегменте памяти).